



KAPITEL 14
Proxies

Mit dem Filterskript aus dem vorigen Kapitel haben wir bereits eine funktionierende Firewall aufgesetzt. Wir können ihre Effizienz allerdings noch steigern, wenn wir zusätzlich zu den Paketfilterregeln auch noch Proxies aufsetzen, die eine Filterung und Protokollierung auf Anwendungsprotokollebene möglich machen.

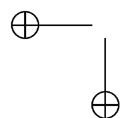
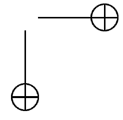
In diesem Kapitel werden Sie daher einige gängige Vertreter dieser Spezies kennenlernen.

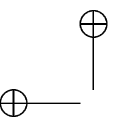
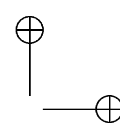
Einrichten eines Web- oder FTP-Proxys

Haben wir ein größeres lokales Netz, in dem mehrere Benutzer regelmäßig im WWW surfen, so wird es vorkommen, daß mehrere von ihnen die gleichen Webseiten betrachten wollen. In diesem Fall empfiehlt es sich, einen cachenden Webproxy zu benutzen. Dieser fordert die betreffende Seite nur beim ersten Mal direkt von dem eigentlichen Server an, alle weiteren Nachfragen werden aus einem Cache bedient.

Darüber hinaus bietet er die Möglichkeit, Zugriffe zu protokollieren und so festzustellen, ob auf Seiten zugegriffen wird, deren Besuch der Policy widerspricht. Insbesondere kann so auch festgestellt werden, ob bestimmte Anwendungen versuchen, eigenmächtig Verbindungen zu Servern ihrer Hersteller aufzubauen. Eine Protokollierung, welcher Anwender bestimmte Zugriffe versucht hat, erfolgt in der hier beschriebenen Konfiguration nicht. Es liegt auch nicht in meiner Absicht, zu beschreiben, wie sie verändert werden müßte, um dies zu erreichen.

Wird die Entscheidung getroffen, der Zugriff auf bestimmte Seiten sei nicht wünschenswert, so bieten die hier betrachteten Proxies die Möglichkeit, diese Zugriffe zu unterbinden. Während dies eine sinnvolle Maßnahme ist, um das Herunterladen von Werbebannern zu unterbinden und Software davon abzuhalten, ihrem Hersteller jeden Start mitzuteilen, ist Filterung als Zensur von Inhalten relativ sinnlos. Wer nach bestimmten Inhalten sucht, wird immer einen Weg finden, zu ihnen zu gelangen. Die Erfahrung zeigt, daß sogar kleine Kinder und relativ unerfahrene Benutzer schnell Wege finden, mißliebige Sperren zu umgehen.

squid

Beim squid handelt es sich um einen cachenden Proxy, der dafür ausgelegt ist, auch große lokale Netze zu bedienen. Bei Bedarf können auch mehrere squids benutzt werden, die sich über ein Protokoll namens ICP gegenseitig befragen, wenn sie eine Seite nicht im eigenen Cache vorrätig haben, um so nur dann eine Verbindung zum eigentlichen Webserver herzustellen, wenn keiner der Proxies eine aktuelle Kopie der Seite besitzt. Die Leistungsfähigkeit des squid scheint so groß zu sein, daß auch einige Internet-Provider ihn einsetzen, was man dann merkt, wenn eine Seite nicht geladen werden kann und eine Fehlermeldung des Proxys angezeigt wird.

Neben der Möglichkeit, mehrfaches Laden derselben Seite zu vermeiden, erlaubt es der squid auch, durch Filtern der HTTP-Header die eigene Privatsphäre zu schützen (siehe Kapitel 4, Abschnitt *Angriffe auf die Privatsphäre*, ab Seite 56). Dabei kann gezielt für jeden HTTP-Header festgelegt werden, ob er erlaubt oder verboten ist.

Schließlich kann der squid auch noch so konfiguriert werden, daß er als transparenter Proxy eingesetzt werden kann.

Grundeinrichtung

Wir haben den squid schon beim Einrichten des Systems installiert. Wird der Proxy beim Start des Rechners nicht automatisch ausgeführt, sollten Sie überprüfen, ob die nötigen Links auf das Runlevel-Skript existieren.

Er kann aber auch manuell mit

```
/.../init.d/squid start
```

gestartet werden. (Bitte ergänzen Sie den Pfad, so wie es für die von Ihnen verwendete Distribution nötig ist.)

Damit ist der squid schon benutzbar. Allerdings gibt es doch einige Einstellungen, die wir in der Datei

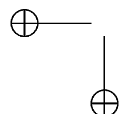
/etc/squid.conf bzw. */etc/squid/squid.conf*

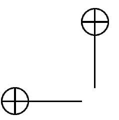
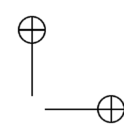
ändern sollten. Zwar ist besagte Datei recht lang, sie ist aber ausgezeichnet kommentiert.

Standardmäßig sind fast alle Einträge auskommentiert, wodurch der squid seine Grundeinstellungen benutzt. Diese wurden zwar sinnvoll gewählt und reichen normalerweise völlig aus, allerdings erfordert unsere Konfiguration in einigen Punkten kleinere Abweichungen. Es empfiehlt sich daher, die Datei umzubenennen und eine neue zu erstellen, die speziell auf die eigenen Bedürfnisse zugeschnitten ist.

Wir beginnen mit den Ports, auf denen squid auf Anfragen wartet:

```
http_port  3128
icp_port   0
htcp_port  0
```

Damit wartet squid auf Port 3128 auf Anfragen. Andere Protokolle, die squid zur Kommunikation mit Nachbarn verwendet, sind abgeschaltet.

Der squid schreibt drei Logdateien. Auch wenn wir ihre Namen und Pfade nicht konfigurieren, werden diese sowohl unter Debian als auch unter SuSE im Verzeichnis */var/log/squid* abgelegt. Wir können diese Einstellung aber auch noch einmal explizit konfigurieren:

```
cache_access_log /var/log/squid/access.log
cache_log        /var/log/squid/cache.log
cache_store_log  /var/log/squid/store.log
```

In der Datei *access.log* protokolliert der squid dabei, welche Anfragen von welchen Rechnern gestellt wurden. Hierbei kann er grundsätzlich zwei Formate verwenden: entweder sein eigenes oder das Common Logfile Format. Wird letzteres gewünscht, so muß

```
emulate_httpd_log on
```

eingetragen werden.

Der squid kennt »ACL« genannte Bezeichner[1], mit denen bestimmte Zugriffe auf Adressen, Protokolle und Zugriffsmethoden spezifiziert werden. Diese werden dann eingesetzt, um bestimmte Zugriffe zu untersagen.

Die folgenden ACLs gehören zu den wenigen Einträgen in der Datei *squid.conf*, die nicht standardmäßig auskommentiert sind:

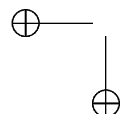
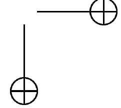
```
acl all src 0.0.0.0/0.0.0.0
acl manager proto cache_object
acl localhost src 127.0.0.1/255.255.255.255
acl to_localhost dst 127.0.0.0/8
acl SSL_ports port 443 563
acl Safe_ports port 80 21 443 563 70 210 1025-65535
acl CONNECT method CONNECT
```

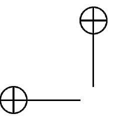
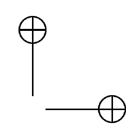
Es kann allerdings sein, daß in der Standardkonfiguration Ihrer Distribution für *SSL_ports* und *Safe_ports* mehr Ports definiert sind. Hier sollten Sie abwägen, ob Sie einen sinnvollen Grund sehen, warum der Proxy Anfragen an diesen Port weiterleitet.

In *Safe_ports* sollten Sie nur solche Ports eintragen, die zu Protokollen gehören, welche vom Proxy unterstützt werden oder die SSL benutzen. SSL-basierte Protokolle müssen zusätzlich noch in *SSL_ports* aufgeführt werden.

Hier sind HTTP (80), FTP (21), Gopher (70), WAIS (210) erlaubt. Zusätzlich sind die SSL-Protokolle HTTPS (443) und NNTPS (563) erlaubt. Daß hier auch alle Ports über 1024 in *Safe_ports* aufgeführt sind, liegt daran, daß dieser Bereich relativ frei genutzt werden kann und es durchaus Webserver gibt, die in diesem Portbereich betrieben werden.

1 Leider benutzt der squid – wie auch viele andere Server – den Begriff ACL (Access Control List), um eine Ressource (oder die Art des Zugriffes darauf) zu beschreiben. An sich ist eine ACL aber mehr. Sie beschreibt neben einer Ressource auch, wer auf sie zugreifen darf.

Um auch den Klienten im lokalen Netz den Zugriff erlauben zu können, brauchen wir aber noch eine weitere ACL:

```
acl Safe_clients src 192.168.20.0/255.255.255.0
```

Nun gilt es zu definieren, welche Zugriffe erlaubt sind. Die folgenden Regeln stammen ebenfalls größtenteils aus der Standardkonfiguration, allerdings wurde auch der lokale Zugriff auf das Cache-Object-Protokoll[2] verboten und der Zugriff für den lokalen Rechner sowie Klienten aus dem lokalen Netz erlaubt:

```
http_access deny manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
http_access allow Safe_clients
http_access allow localhost
http_access deny all
```

Ein Wort noch zu der ACL `CONNECT`: Diese definiert einen Zugriff über die HTTP-Methode `CONNECT`, die benutzt wird, wenn eine SSL-Verbindung aufgebaut werden soll. Da SSL-Verbindungen verschlüsselt sind, kann der Proxy ihren Inhalt weder kontrollieren, noch darf er ihn verändern. Deswegen wird er mit dieser Methode angewiesen, eine Verbindung zu einem angegebenen Port eines Zielrechners zu öffnen und die dann folgenden Daten völlig transparent weiterzuleiten.

Ohne die obigen Regeln könnte man die `CONNECT`-Methode dazu benutzen, Filterregeln im Proxy für beliebige Verbindungen außer Kraft zu setzen. Es existieren bereits frei verfügbare Programme, die diesen Mechanismus dazu benutzen, Schutzmechanismen in Firewalls auszuhebeln. In unserem Fall ist `CONNECT` daher nur für die Ports 443 (HTTPS) und 563 (NNTPS) zulässig.

Wir können auch konfigurieren, mit welchen Rechten der `squid` arbeiten soll. Dazu existieren die Optionen `cache_effective_user` und `cache_effective_group`. Unter Debian und SuSE reicht es aber, wenn diese Optionen auskommentiert sind. In diesem Fall werden unter SuSE der Benutzer `squid` und die Gruppe `nobody` verwendet, während unter Debian sowohl als Benutzer wie auch als Gruppe `proxy` verwendet wird.

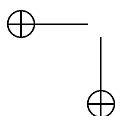
Wollen wir dies explizit konfigurieren, so sieht das unter SuSE folgendermaßen aus:

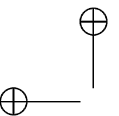
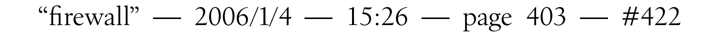
```
cache_effective_user squid
cache_effective_group nobody
```

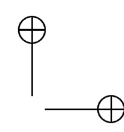
Unter Debian dagegen:

```
cache_effective_user proxy
cache_effective_group proxy
```

2 Erlaubt es, Statistiken über die Webzugriffe abzufragen. Zum `squid` gehört auch ein cgi-Skript, das es erlaubt, dies komfortabel über einen Webbrowser zu tun. Allerdings macht es keinen Sinn, auf der Firewall extra zu diesem Zweck einen Webserver aufzusetzen. Darüber hinaus wäre ein Zugriff über das cgi-Skript aus Sicht des `squid` immer lokal, womit eine Einschränkung von Zugriffen nicht mehr möglich ist.





In den Standardeinstellungen überträgt der `squid` für jede Anfrage die IP-Adresse des anfragenden Rechners. Damit geben wir fremden Rechnern Informationen über die Struktur unseres lokalen Netzes. Wir schalten dieses Feature daher ab:

```
forwarded_for    off
```

Der `squid` kann Statistiken über SNMP zur Verfügung stellen. Das wollen wir nicht[3]:

```
snmp_port        0
```

Schutz der Privatsphäre

Der `squid` kann aber noch mehr als Webseiten cachen. Jede HTTP-Nachricht enthält neben der eigentlichen Anfrage beispielsweise nach einer Webseite auch noch diverse Header mit zusätzlichen Informationen. Dabei kann z. B. angegeben werden, welchen Browser der Surfer benutzt, welche Webseite er zuletzt besichtigt hat – ja, es ist sogar ein Header für die E-Mail-Adresse des Benutzers vorgesehen. Auch die berüchtigten Cookies sind als HTTP-Header realisiert.

Der `squid` kann diese Header filtern.

Die Filterregeln haben dabei den folgenden Aufbau:

```
header_access <header_name> allow|deny [!]<aclname>
```

Gibt man als *<header_name>* das Schlüsselwort `ALL` an, so wird die Regel auf alle Header angewendet.

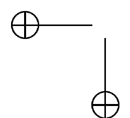
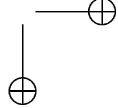
Will man nun explizit bestimmte Header unterdrücken, kann dies folgendermaßen geschehen:

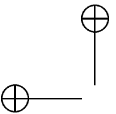
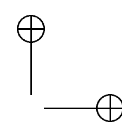
```
header_access From deny all
header_access Referer deny all
header_access Server deny all
header_access User-Agent deny all
header_access WWW-Authenticate deny all
header_access Link deny all
```

Den umgekehrten Fall könnte man dagegen so ausdrücken:

```
header_access Allow allow all
header_access Authorization allow all
header_access WWW-Authenticate allow all
header_access Cache-Control allow all
header_access Content-Encoding allow all
header_access Content-Length allow all
header_access Content-Type allow all
header_access Date allow all
header_access Expires allow all
header_access Host allow all
```

3 Früher hat man `-1` statt `0` verwendet.

```
header_access If-Modified-Since allow all
header_access Last-Modified allow all
header_access Location allow all
header_access Pragma allow all
header_access Accept allow all
header_access Accept-Charset allow all
header_access Accept-Encoding allow all
header_access Accept-Language allow all
header_access Content-Language allow all
header_access Mime-Version allow all
header_access Retry-After allow all
header_access Title allow all
header_access Connection allow all
header_access Proxy-Connection allow all
header_access All deny all
```

Beide Regelwerke filtern den Namen des Webbrowsers. Da einige Server diesen Header aber zwingend verlangen, bietet der squid die Möglichkeit, einen eigenen Wert für diesen Header einzutragen. In der Beispieldatei der Version 2.4 wird dafür

```
fake_user_agent  Nutscrape/1.0 (CP/M; 8-bit)
```

vorgeschlagen, während die gleiche Einstellung in Version 2.5 so aussieht:

```
header_replace User-Agent Nutscrape/1.0 (CP/M; 8-bit)
```

Zusammenarbeit mit anderen Proxies auf der Firewall

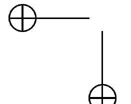
Einen squid kann man prinzipiell in ein kompliziertes Geflecht von Geschwister- und Eltern-Caches einbinden, über das der Proxy erst bei seinen »Familienmitgliedern« nachfragt, bevor er versucht, eine Seite direkt zu laden. Für den Gebrauch in unserem Szenario ist dies aber nicht nötig. Allerdings kann es sinnvoll sein, auf der Firewall einen zweiten Proxy zu installieren, der die Fähigkeiten des squid ergänzt. So wäre es z. B. möglich, mit einem weiteren Proxy aktive Inhalte (Java, JavaScript ...) aus den heruntergeladenen Dokumenten zu filtern, während der squid für das Cachen und Filtern bestimmter Header zuständig ist.

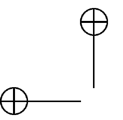
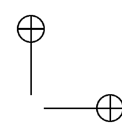
Nehmen wir also einmal an, unser Rechner hieße *fw.local.dom*. Alle Anfragen sollen nicht direkt, sondern über einen Proxy auf Port 8080 gestellt werden. Anfragen an Rechner im lokalen Netz werden direkt beantwortet. Dies ergäbe folgende Konfiguration:

```
cache_peer fw.local.dom parent 8080 0 default
acl local-servers dstdomain local.dom
acl all src 0.0.0.0/0.0.0.0
never_direct deny local-servers
never_direct allow all
```

Es scheint allerdings Fälle zu geben, in denen Proxies keine Verbindung öffnen können, wenn sie Anfragen von der Adresse 127.0.0.1 erhalten. In diesem Fall kann es nötig sein, die Anfrage statt dessen von der internen Adresse der Firewall aus zu stellen[4]:

4 Dieses Problem hatte ich persönlich einmal mit einer Version des http-gw.



```
tcp_outgoing_address 192.168.20.15
```

Der squid als transparenter Proxy

Soll der squid auch wie in Kapitel 11, Unterabschnitt *Transparente Proxies*, ab Seite 291 beschrieben als transparenter HTTP-Proxy eingesetzt werden, so muß er gegebenenfalls den Host-Header auswerten, wenn eine URL unvollständig ist. Dies wird mit den folgenden Anweisungen konfiguriert:

```
httpd_accel_host virtual
httpd_accel_port 80
httpd_accel_with_proxy on
httpd_accel_uses_host_header on
```

Automatischer Start

Unter Debian wird ein installierter squid automatisch beim Booten gestartet. Unter SuSE muß der Dienst erst registriert werden:

```
# insserv squid
```

Wollen wir nicht bis zum nächsten Bootvorgang warten, so können wir den Dienst auch manuell starten:

```
# /etc/init.d/squid start
```

Ist er dagegen bereits aktiv und wir wollen ihn nur dazu bringen, seine Konfiguration erneut einzulesen, so können wir den folgenden Befehl benutzen:

```
# /etc/init.d/squid reload
```

Alternativ können wir ihn auch beenden und neu starten lassen:

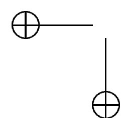
```
# /etc/init.d/squid restart
```

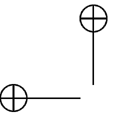
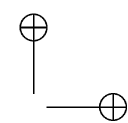
Privoxy

Der Privoxy ist der direkte Nachfolger des bekannten Internet Junkbusters. Nachdem der Internet Junkbuster vom Hersteller nicht mehr weiterentwickelt wurde, nahmen Freiwillige es auf sich, den unter GPL stehenden Code weiterzuentwickeln.

Aus diesem Grunde sind die Grundeigenschaften des Privoxy und des Junkbusters immer noch recht ähnlich. Auch der Privoxy cacht einmal heruntergeladene Seiten nicht, und die Benutzung als transparenter Proxy ist ebenfalls nicht möglich.

In anderer Hinsicht hat er sich gegenüber dem Junkbuster allerdings gravierend verbessert:



- Es werden nicht mehr nur URLs, sondern auch Inhalte von Webseiten gefiltert. Dies ermöglicht es, z. B. Pop-ups, Webbugs, Graphiken, deren Größe auf ein Werbebanner hindeutet, sowie diverse JavaScript-Ärgernisse zu filtern.
- Filter sind nicht mehr fest im Programm einkompiliert, sondern können in Form regulärer Ausdrücke in Konfigurationsdateien definiert werden.
- Es wird zwischen allgemeinen und benutzerdefinierten Regeln unterschieden. Damit ist es möglich, regelmäßig aktuelle Regeln von der Webseite des Projekts herunterzuladen, ohne bei deren Installation selbst definierte Regeln zu überschreiben.
- Es ist möglich, ihn in einem `chroot`-Käfig zu betreiben. Dateien außerhalb dieses Käfigs sind für ihn dann nicht mehr zugreifbar.
- Obwohl er standardmäßig dasselbe Logformat benutzt wie der `junkbuster`, kann der Privoxy so konfiguriert werden, daß er statt dessen das Common Logfile Format benutzt. Dieses ist in einer Mehrbenutzerumgebung deutlich aussagekräftiger, da die Einträge die IP-Adresse des Klienten enthalten.

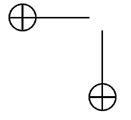
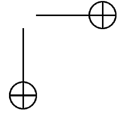
All dies macht ihn deutlich vielseitiger und erlaubt es sehr genau zu steuern, womit man belästigt werden will. Allerdings erkennt man an den Details, daß der Hauptfokus der Entwickler darauf liegt, einen Dienst zu entwickeln, den der einzelne Benutzer lokal bei sich auf dem Rechner installiert. So kann man den Proxy z. B. über ein Webinterface konfigurieren. Aktiviert man dieses Feature aber, so darf es von jedem Benutzer der Firewall aufgerufen werden, und sämtliche Änderungen wirken sich auf alle Benutzer des Systems aus. Sobald mehrere Benutzer über die Firewall surfen, besteht mithin die Gefahr, daß die Konfiguration nach kurzer Zeit in einem völlig zufälligen Zustand ist.

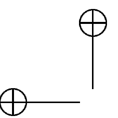
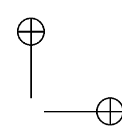
Auch bewirkt die Konfiguration von Filtern, daß die Webseite komplett in den Hauptspeicher geladen wird. Während Webseiten selber als reine Textdateien vermutlich noch in den Hauptspeicher eines halbwegs aktuellen Rechners passen, sieht das bei der Filterung von Bildern schon anders aus. Standardmäßig ist ein Feature aktiviert, das animierte GIFs in Standbilder verwandelt. Auch dieses Feature bewirkt, daß jedes Bild zur Untersuchung erst einmal komplett in den Hauptspeicher geladen wird.

Diese und andere Details lassen schon Fragen aufkommen, inwieweit die Software für den Betrieb mit mehr als einer Handvoll Benutzern wirklich geeignet ist. Andererseits sind die Filtermöglichkeiten dieses Proxys wirklich herausragend, weswegen er in kleineren Netzen eindeutig eine Betrachtung wert ist.

Einrichten eines chroot-Käfigs

Sicherheitshalber wollen wir den Privoxy in einen Käfig sperren, der aus dem Heimatverzeichnis des Benutzers `privoxy` besteht. Alle Dateien außerhalb dieses Verzeichnisses sind aus seiner Sicht nicht mehr vorhanden. Auch wenn es einem Angreifer gelingt, den Dienst zu korrumpieren und er den Proxy dazu bringt, beliebige Befehle auszuführen, so bleibt er doch auf das Verzeichnis */var/lib/privoxy* beschränkt und kann außerhalb dieses Käfigs keinen Schaden anrichten.

Unter SuSE 9.3 ist solch ein Käfig bereits vorhanden, und die in diesem Abschnitt beschriebenen Arbeiten entfallen. Unter Debain 3.1 ist der chroot-Betrieb dagegen nicht vorgesehen, so daß wir ein paar Vorarbeiten leisten müssen.

Wir beginnen damit, daß wir ein Verzeichnis anlegen, das uns später als Käfig dienen soll:

```
# mkdir -p -m0755 /var/lib/privoxy
```

Damit ein Ausbrechen aus unserem Käfig nicht möglich ist, darf der Dienst nicht mit root-Rechten laufen. Wir benötigen also einen eigenen Benutzer für den Dienst. Unter Debian ist hierfür der Benutzer privoxy vorgesehen. Allerdings ist für diesen Benutzer die Gruppe nogroup als Standardgruppe eingetragen. Wir wollen ihm im folgenden eine eigene Gruppe geben, die er mit keinem anderen Dienst teilt:

```
# groupadd privoxy
```

Nun können wir ihm die Gruppe zuweisen und ihm das Käfig-Verzeichnis als Heimatverzeichnis eintragen:

```
# usermod -c "Privoxy Daemon" \
> -d /var/lib/privoxy \
> -g privoxy \
> -G '' \
> -s /bin/false \
> privoxy
```

Nun müssen wir unter */var/lib/privoxy* das normale Dateisystem nachbilden. Der privoxy muß dort alle Dateien vorfinden, die er für den Betrieb braucht. Als erstes erstellen wir das Verzeichnis für die Konfigurationsdateien und kopieren die Dateien aus dem Originalverzeichnis hinein:

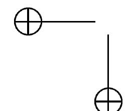
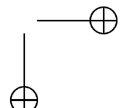
```
# cd /var/lib/privoxy
# mkdir -m0755 etc
# cp -Rdfp /etc/privoxy etc/privoxy
```

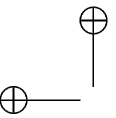
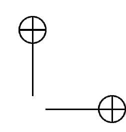
Es existieren auch bereits Logdateien für den Dienst. Diese müssen wir ebenfalls kopieren:

```
# mkdir -m0755 -p var/log/
# cp -Rdfp /var/log/privoxy var/log/privoxy
```

Als nächstes benötigt der Privoxy noch einige Systemdateien. Die folgenden Dateien werden zur Auflösung von Netzwerknamen benötigt:

```
# cp -dfp /etc/resolv.conf etc
# cp -dfp /etc/hosts etc
# cp -dfp /etc/services etc
# cp -dfp /etc/protocols etc
```

Damit die Namensauflösung aber auch wirklich funktioniert, werden noch einige Bibliotheken benötigt, die erst geladen werden, wenn das erste Mal versucht wird, einen Rechnernamen oder eine IP-Adresse aufzulösen:

```
# mkdir -m0755 lib
# cp -dfp /lib/ld* lib
# cp -dfp /lib/libnss_dns* lib
# cp -dfp /lib/libnss_files* lib
# cp -dfp /lib/libresolv* lib
```

Schließlich können wir dem Proxy noch mitteilen, in welcher Zeitzone er sich befindet. Dies spielt insbesondere dann eine Rolle, wenn wir ihn Logdateien schreiben lassen. Ohne die folgenden Einstellungen wären alle Einträge in UTC.

Konfiguriert wird die Zeitzone über die Datei */etc/localtime*. In einigen Systemen ist dies eine normale Datei, in anderen ein symbolischer Link:

```
# ls -l /etc/localtime
lrwxrwxrwx  1 root  root   33 Feb 17 21:26 /etc/localtime ->
/usr/share/zoneinfo/Europe/Berlin
```

Hier handelt es sich um einen Link nach */usr/share/zoneinfo*. Wir kopieren also als erstes den Link, wobei wir darauf achten, `cp` mit dem Parameter `-d` anzuweisen, den Link als Link zu kopieren und nicht etwa aufzulösen:

```
# cp -dfp /etc/localtime etc
```

Nun müssen wir nur noch die eigentliche Datei kopieren:

```
# mkdir -p -m0755 usr/share/zoneinfo/Europe
# cp -dfp /usr/share/zoneinfo/Europe/Berlin usr/share/zoneinfo/Europe/
```

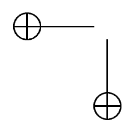
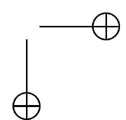
Nun sollte der Privoxy alle Dateien vorfinden, die er benötigt. Die Konfigurations- und Logdateien außerhalb seines Käfigs werden eigentlich nicht mehr benötigt. Prinzipiell spricht nichts dagegen, sie zu löschen:

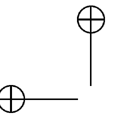
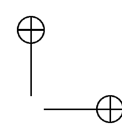
```
# rm -rf /etc/privoxy
# rm -rf /var/log/privoxy
```

Nun müssen wir noch das Runlevel-Skript anpassen, damit der `privoxy` auch beim Systemstart automatisch aufgefordert wird, sich in seinem `chroot`-Käfig einzuschließen. Dazu editieren wir das Skript `/etc/init.d/privoxy`. In ihm finden wir folgende Zeilen:

```
case "$1" in
 start)
  echo -n "Starting $DESC: "
  start-stop-daemon --oknodo --start --quiet --pidfile $PIDFILE \
  --exec $DAEMON -- --pidfile $PIDFILE --user $OWNER $CONFIGFILE \
  2>> /var/log/privoxy/errorfile
```

Alle Argumente, die `start-stop-daemon` hinter `--` übergeben werden, werden dabei unverändert an den `privoxy` weitergeleitet. Wir müsen dort also nur die Option `--chroot` einfügen:


```
case "$1" in
 start)
  echo -n "Starting $DESC: "
  start-stop-daemon --oknodo --start --quiet --pidfile $PIDFILE \
  --exec $DAEMON -- --pidfile $PIDFILE --user $OWNER --chroot \
  $CONFIGFILE 2>> /var/log/privoxy/errorfile
```

Konfiguration

Nachdem wir die Software installiert haben, müssen wir sie noch konfigurieren. Die Standardeinstellungen sind zwar schon recht brauchbar, im einen oder anderen Fall kann man sie aber noch optimieren.

Ich kann hier allerdings nicht auf jedes Detail eingehen. Die einzelnen Dateien sind aber recht gut kommentiert, und das Benutzerhandbuch, das Sie in HTML-Form unter */usr/share/doc/privoxy/manual* finden, ist ebenfalls recht ausführlich. Aus diesem Grund werde ich im folgenden nur auf die Einstellungen eingehen, bei denen ich ein Abgehen von den Standardeinstellungen für sinnvoll halte.

Beginnen wir mit der Datei */var/lib/privoxy/etc/config*. Sie regelt grundsätzliche Dinge wie z. B. Pfade zu anderen Dateien, wie ausführlich protokolliert wird oder auf welcher Adresse der Proxy Anfragen entgegennimmt.

Die grundlegenden Pfade und Dateien entsprechen der Konfiguration, die wir vor dem Kompilieren des Servers vorgenommen haben. Interessant ist daher nur die folgende Datei:

```
jarfile jarfile
```

Das Verzeichnis, in dem sich die Datei befindet, wurde mit einer anderen Option angegeben.

Das `jarfile` enthält alle Cookies, die der Privoxy abgefangen hat. Diese Speicherung ist normalerweise aber eher überflüssig.

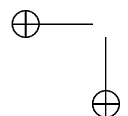
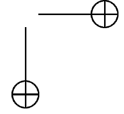
Es bietet sich daher an, die Option auszukommentieren und auf die Protokollierung zu verzichten:

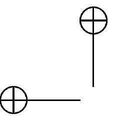
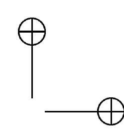
```
#jarfile jarfile
```

Die nächste wichtige Option ist `debug`. Sie regelt, was in `logfile` protokolliert wird. Die Standardvorgaben sind:

```
debug   1    # show each GET/POST/CONNECT request
debug   4096 # Startup banner and warnings
debug   8192 # Errors - *we highly recommended enabling this*
```

Auf diese Weise werden alle Zugriffe sowie Meldungen beim Start des Proxys und Fehler protokolliert. Das sieht in etwa so aus:

```
Jun 08 17:31:53 Privoxy(01024) Info: Privoxy version 3.0.3
Jun 08 17:31:53 Privoxy(01024) Info: Program name: privoxy
Jun 08 17:31:54 Privoxy(01024) Info: Listening on port 8118 for local conn
ections only
[...]
Jun 08 18:58:20 Privoxy(01026) Request: www.dilbert.com/comics/dilbert/arc
hive/dilbert-20040606.html
[...]
Jun 08 18:58:41 Privoxy(04100) Request: c4.maxserving.com/iserver/site=531
1/area=homepage/aamsz=banner/PageID=1234567 crunch!
```

Wir sehen also, der Proxy startet, ein Benutzer ruft einen Webcomic auf, und ein Werbebanner wird unterdrückt. Daß das Werbebanner zu dem Webcomic gehört, ist aus dem Protokoll nicht ersichtlich. Wir können auch nicht erkennen, welcher unserer Benutzer hier eine Seite angefordert hat. Sobald mehr als ein Benutzer den Proxy verwendet oder mehr als eine Seite gleichzeitig aufgerufen wird, wird das Protokoll unauswertbar.

Als Alternative bietet sich das Common Logfile Format an. Es wird mit der folgenden Zeile aktiviert:

```
debug       512 # Common Log Format
```

Die gleichen Zugriffe sähen damit folgendermaßen aus:

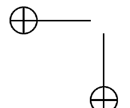
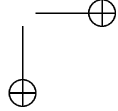
```
127.0.0.1 - - [08/Jun/2004:19:05:54 +0200] "GET http://www.dilbert.com/comics/di
lbert/archive/dilbert-20040606.html HTTP/1.0" 200 45619
[...]
127.0.0.1 - - [08/Jun/2004:19:05:59 +0200] "GET http://c4.maxserving.com/iserver
/site=5311/area=homepage/aamsz=banner/PageID=1234567 HTTP/1.0" 200 3
```

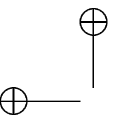
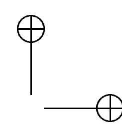
Hier haben wir als zusätzliche Information die IP-Adresse des Klienten, über die wir mit etwas Glück den Benutzer zuordnen können. Andererseits fehlt uns hier die Information, ob Inhalte geblockt wurden.

Wir können grundsätzlich auch alle Möglichkeiten für `debug` kombinieren, aber das Ergebnis wird schnell unleserlich, insbesondere da das Common Logfile Format und das ansonsten benutzte Syslog-Format sehr stark voneinander abweichen und jeder Zugriff doppelt protokolliert wird:

```
Jun 08 19:18:25 Privoxy(48130) Request: www.dilbert.com/comics/dilbert/archive/d
ilbert-20040606.html
127.0.0.1 - - [08/Jun/2004:19:18:39 +0200] "GET http://www.dilbert.com/comics/di
lbert/archive/dilbert-20040606.html HTTP/1.0" 200 45619
[...]
Jun 08 19:18:43 Privoxy(51204) Request: c4.maxserving.com/iserver/site=5311/area
=homepage/aamsz=banner/PageID=1234567 crunch!
127.0.0.1 - - [08/Jun/2004:19:18:43 +0200] "GET http://c4.maxserving.com/iserver
/site=5311/area=homepage/aamsz=banner/PageID=1234567 HTTP/1.0" 200 3
```

Oft sind auch ganze Gruppen von Zugriffen erst einmal in der einen Form und dann noch einmal als Gruppe im anderen Format protokolliert, da der Eintrag im Common Logfile Format teilweise erst mehrere Sekunden nach dem Eintrag im Syslog-Format er-

folgt. Damit lassen sich die einzelnen Zugriffseinträge in den unterschiedlichen Formaten nicht wirklich gut einander zuordnen.

Das Handbuch empfiehlt daher, entweder die Standardeinstellungen oder das Common Logfile Format zu benutzen, nicht aber beides gemeinsam.

Ist kein `debug`-Eintrag vorhanden, dann findet keine Protokollierung statt.

Eine weitere Einstellung gibt an, auf welcher Adresse der Proxy Anfragen beantwortet. Standardmäßig ist hier *127.0.0.1* eingetragen, womit Zugriffe nur von der Firewall selbst aus möglich sind. Hier sollten wir die IP-Adresse unseres internen Netzwerk-Interfaces angeben, um den Klienten im LAN die Nutzung des Proxys zu ermöglichen:

```
listen-address  192.168.20.15:8118
```

Wie bereits erwähnt, läßt sich der Proxy über sein Web-Interface auch fernkonfigurieren. Wir haben diese Funktionen aber abgestellt. In der gegenwärtigen Version werden die entsprechenden Schaltflächen aber trotzdem angezeigt. Wir sollten also auch noch einmal in der Konfiguration festhalten, daß die Funktionen nicht zulässig sind:

```
enable-remote-toggle  0
[...]
enable-edit-actions 0
```

Der Proxy bietet auch die Möglichkeit, dediziert festzulegen, welche Rechner auf ihn zugreifen dürfen und welche nicht. Dazu definiert man zuerst mit `permit-access`, welche Rechner ihn benutzen dürfen, um dann mit `deny-access` die Rechner auszuschließen, die das nicht können sollen. Grundsätzlich können mehrere Zeilen mit `permit-access`- und `deny-access`-Einträgen vorkommen. Passen im konkreten Fall mehrere Zeilen zu dem anfragenden Klienten, so wird die jeweils letzte genommen.

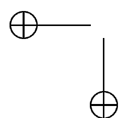
Wollen wir z. B. prinzipiell allen Rechnern im LAN den Zugriff ermöglichen, aber einen speziellen Rechner *192.168.20.13* explizit davon ausnehmen, so können wir dies folgendermaßen konfigurieren:

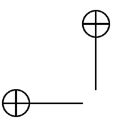
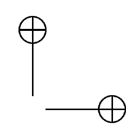
```
permit-access  192.168.20.0/24
deny-access    192.168.20.13
```

Wollen wir Anfragen nicht selber bearbeiten, sondern sie an einen anderen Proxy weiterleiten, so kann dies mit der `forward`-Option geschehen. Wie schon zuvor sind mehrere `forward`-Zeilen möglich. Es greift immer die letzte passende Zeile. Die Zeilen haben den folgenden Aufbau:

```
forward [<Rechner>][/[<Pfad>]] <Parent>
```

<*Rechner*> kann auch eine partielle Angabe sein, z. B. `.oreilly.de`, wodurch alle Rechneradressen gemeint sind, die auf `.oreilly.de` enden. Ein `/` alleine steht für beliebige Rechner und Pfade. Pfade können reguläre Ausdrücke enthalten. Schließlich kann auch mit `:`<*Port*> statt bestimmter Adressen vorgegeben werden, daß Zugriffe auf bestimmte Ports abgeglichen werden sollen.



Der <*Parent*> hat die Form <*DNS-Name*>[:<*Port*>]. Wird statt dessen nur ein Punkt angegeben, so findet keine Weiterleitung statt.

Im folgenden Beispiel wollen wir alle Anfragen an einen Proxy auf dem gleichen Rechner weiterleiten, der auf Port 8080 auf Anfragen wartet. Dieser kann aber kein SSL, weshalb wir solche Anfragen direkt ausführen:

```
forward /       192.168.20.15:8080
forward :443    .
```

Start und Funktionstests

Nun können wir den Privoxy starten:

```
# /etc/init.d/privoxy start
```

Um nun zu testen, ob der Privoxy auch arbeitet, können wir sein Webinterface aufrufen, indem wir mit dem `lynx` die URL *http://config.privoxy.org* oder kurz *http://p.p* aufrufen.

Damit unser Browser auch den Privoxy benutzt, müssen wir ihm dessen Adresse mitteilen. Dies geschieht über die Environment-Variable *http_proxy*:

```
> export http_proxy='http://localhost:8118'
> lynx http://p.p
```

Unser Browser sollte uns nun eine Seite anzeigen, über die wir die aktuelle Konfiguration abfragen können, wo uns Informationen über die Header angezeigt werden, die unser Browser mit jeder Anfrage sendet und auf der wir für eine bestimmte URL herausfinden können, welche Regeln für sie greifen:

```
                                                    Privoxy@localhost (p1 of 2)

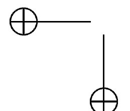
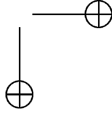
This is Privoxy 3.0.3 on localhost (127.0.0.1), port 8118

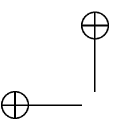
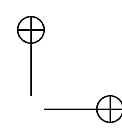
Privoxy Menu:

     * View the current configuration
     * View the source code version numbers
     * View the request headers.
     * Look up which actions apply to a URL and why
     * Documentation

Support and Service via Sourceforge:

   We value your feedback. To provide you with the best support, we ask
   that you:
     * use the support forum to get help.
     * submit banners and all problems with the actions file only through
       the actions file feedback system.
     * submit bugs only through our bug tracker. Make sure that the bug
       has not yet been submitted.
-- press space for next page --
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go back.
 H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

Nachdem der Start nun funktioniert, können wir das Runlevel-Skript verlinken, und der Proxy wird in Zukunft bei jedem Systemstart aktiviert.

Tuning der Filter

Wir können den Privoxy nun nutzen. Seine voreingestellten Filterregeln sind im großen und ganzen sinnvoll eingestellt und filtern den Großteil der Werbebanner, denen man beim Surfen begegnet. Auch für den Schutz der Privatsphäre und vor einer Reihe von Ärgernissen, denen man beim Surfen begegnet, ist gesorgt.

Standardmäßig werden z. B.

- animierte GIFs zu statischen Bildern konvertiert,
- Webseiten so geändert, daß sie Kind-Fenster nicht im Kiosk-Modus, das heißt ohne Adreß- und Menü-Leiste öffnen,
- mit JavaScript geöffnete Pop-up-Fenster unterdrückt, die der Benutzer nicht explizit angefordert hat,
- Bilder gefiltert, deren Größe typisch für Werbebanner ist,
- Webbugs gefiltert, d. h. 1 × 1 Pixel große Graphiken, die dazu dienen, das Surfverhalten auszuspionieren,
- JavaScript-Befehle gefiltert, die bewirken, daß Fenster sich von selbst bewegen oder ihre Größe verändern,
- Referer-Header so umgeschrieben, daß die vorher besuchte Seite immer auf dem Server zu liegen scheint, den man im Moment besucht[5],
- Cookies so verändert, daß der Browser sie am Ende der Sitzung löscht.

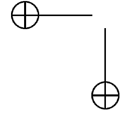
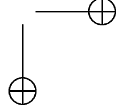
Irgendwann werden Sie aber auf Seiten stoßen, auf denen diese Filterung zu Problemen führt. Oder Sie stellen fest, daß Werbebanner von den Filterregeln übersehen werden. In diesem Fall werden Sie wahrscheinlich den Wunsch verspüren, eigene Regeln zu definieren, die das Problem beheben.

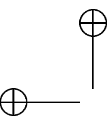
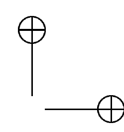
Der Privoxy bietet hierzu recht komplexe Filtermöglichkeiten an. Diese hier komplett aufzuführen, würde leider den verfügbaren Platz übersteigen. Durch die Verwendung regulärer Ausdrücke kann man Webseiten fast beliebig umgestalten.

Ich werde mich hier daher auf die Aufstellung einfacher Regeln für die Lösung von Grundproblemen beschränken. Als weiterführende Lektüre empfehle ich Ihnen das Benutzerhandbuch zu Privoxy, das Sie von der Privoxy-Webseite als PDF-Datei herunterladen können. Es wurde auch als Sammlung von HTML-Seiten unter

/usr/share/doc/privoxy

5 Dieses Fälschen des Referer-Headers ist wirkungsvoller, als ihn einfach zu unterdrücken. Es unterbindet genauso effektiv das Ausspionieren von Surfgewohnheiten mittels Werbebannern und Webbugs, während man trotzdem noch auf die meisten Seiten kommt, die den Referer-Header überprüfen, um zu testen, ob jemand Ihre Inhalte in fremde Seiten einbaut.

installiert. Auch die Konfigurationsdateien sind ausgezeichnet kommentiert. Die Kommentare entsprechen teilweise den zugehörigen Passagen im Benutzerhandbuch.

Nach diesem Hinweis lassen Sie uns nun mit der Konfiguration beginnen. Der Privoxy kennt sogenannte *Action Files*. Diese enthalten Regeln, die der Reihe nach daraufhin überprüft werden, ob sie zu der URL einer angeforderten Seite passen. Ist dies der Fall, so wird vermerkt, daß bestimmte Aktionen auszuführen sind. Spätere Regeln können frühere Regeln ganz oder teilweise aufheben. Im Zweifel zieht immer die zuletzt gelesene Regel.

Am Ende wird überprüft, welche Aktionen tatsächlich ausgeführt werden sollen. Diese Aktionen können entweder eingebaut sein, oder es handelt sich um sogenannte *Filter*, das heißt reguläre Ausdrücke, die Texte in der Webseite ersetzen. Filter werden aus einer Konfigurationsdatei gelesen und sind prinzipiell frei konfigurierbar.

Die bereitgestellten Filter reichen aber für unsere Zwecke vollkommen, weshalb wir uns den Action Files zuwenden. Standardmäßig existieren derer zwei:

<**chroot-Käfig**>*/etc/privoxy/default.action* Diese Datei enthält Grundeinstellungen, die für alle URLs gelten, und spezielle Regeln, die dazu dienen, URLs zu filtern, die auf Werbebanner verweisen. Die Regeln werden ständig weiterentwickelt. Es empfiehlt sich durchaus, diese Datei regelmäßig zu aktualisieren.

<**chroot-Käfig**>*/etc/privoxy/user.action* In dieser Datei können Sie eigene Sonderregeln für die URLs definieren, die von den Standardregeln nicht angemessen behandelt werden.

Obwohl die Datei *default.action* recht umfangreich ist, interessiert uns an ihr eigentlich nur der erste Abschnitt. Abschnitte haben die folgende Form:

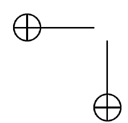
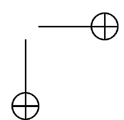
```
{ <Aktion1> <Aktion2> ...}
<URL-Muster1>
<URL-Muster2>
...
```

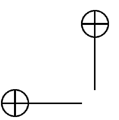
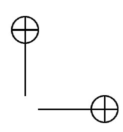
Der erste Abschnitt enthält etwa 40 Aktionen. Da diese nicht auf eine Zeile passen, werden die Zeilenenden durch ein unmittelbar davorstehendes \ auskommentiert:

```
{ \
<Aktion1> \
<Aktion2> \
...}
<URL-Muster1>
<URL-Muster2>
...
```

Das URL-Muster des ersten Abschnittes ist ein einzelnes `/`. Dieses Muster paßt auf alle URLs. Aktionen, die in diesem Abschnitt definiert werden, werden also immer ausgeführt, es sei denn, eine spätere Regel ändert dies.

Wenn wir also eine der eingestellten Filterregeln ablehnen und generell aktivieren oder abschalten wollen, so müssen wir hier vor der jeweiligen Aktion das `-` in ein `+` ändern

oder umgekehrt, je nachdem, was dort eingetragen ist. Ein + aktiviert einen Filter, während - ihn abschaltet.

Wollen wir z. B. auf jeder Webseite das Wort `Microsoft` in `MicroSuck` umsetzen, so brauchen wir nur den Filter `fun` zu aktivieren, indem wir

```
-filter{fun}
```

in

```
+filter{fun}
```

ändern. Der Filter spielt außerdem noch »Buzzword Bingo«, indem er jedes Vorkommen einer Reihe von Marketing-Ausdrücken durch `Bingo!` in leuchtend roten Buchstaben ergänzt.

In der Regel haben wir aber eher Probleme mit speziellen Webseiten. Für diese sollten wir dann auch spezielle Regeln definieren, die nur für diese Seiten gelten. Dies geschieht in der Datei *user.action*.

Um diese nutzen zu können, müssen wir einige der elementaren Aktionen kennen, die der Privoxy benutzt:

block URLs, auf die diese Aktion angewendet wird, werden nicht heruntergeladen, sondern es wird eine spezielle Seite angezeigt, die erklärt, daß die Seite gesperrt ist. Je nachdem, mit welchen Einstellungen der Proxy kompiliert wurde, wird außerdem ein Link angezeigt, der es einem erlaubt, die Seite trotzdem zu laden.

Wenn auf die Seite auch die Aktion `handle-as-image` angewendet wird, so wird statt eines Textes eine Graphik angezeigt. Vorgabe ist in diesem Fall ein Schachbrettmuster. Man kann aber mit `set-image-blocker` die Vorgabe ändern.

crunch-incoming-cookies Server können keine Cookies setzen.

crunch-outgoing-cookies Einmal gesetzte Cookies werden nicht an den Server zurückgesendet.

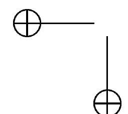
deanimategifs{<Modus>} gibt an, daß animierte GIF-Graphiken nur statisch angezeigt werden sollen. *<Modus>* kann dabei entweder `first` oder `last` sein. Im ersten Fall wird das erste der Animation angezeigt, im zweiten das letzte. Voreingestellt ist, für alle animierten GIF-Bilder nur das letzte Bild anzuzeigen.

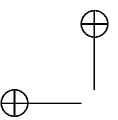
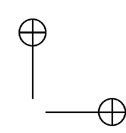
filter{<Filter>} aktiviert den Filter *<Filter>*. `-filter` ohne Argumente deaktiviert alle Filter für die angegebenen URLs. Eine Liste aller Filter finden Sie in der Benutzerdokumentation. Hier ein paar Beispiele:

content-cookies filtert Cookies, die über HTML-Tags oder JavaScript-Befehle gesetzt werden.

unsolicitated-popups filtert alle Pop-up-Fenster, die beim Laden der Seite geöffnet werden. Fenster, die erst durch eine Handlung des Benutzers geöffnet werden, werden davon nicht erfaßt.

all-popups filtert alle Pop-up-Fenster.



handle-as-image erklärt das Objekt, auf das die URL verweist, zum Bild (vgl. `block`). Standardmäßig wird diese Aktion für Dateien ausgeführt, die die folgenden Endungen haben: `gif`, `jpeg`, `jpg`, `png`, `bmp` und `ico`.

kill-popups ist eine veraltete Form, Pop-up-Fenster zu unterdrücken. Heute verwendet man die oben aufgeführten Filter.

limit-connect{<Ports>} erlaubt nur CONNECT-Anfragen an die angegebenen Ports. CONNECT-Anfragen werden für SSL genutzt. Dabei werden alle Daten weitergereicht, ohne daß der Proxy diese in irgendeiner Weise verändert. Das kann dazu benutzt werden, die Filterung zu umgehen. Standardmäßig ist daher nur Port 443 erlaubt.

Diese Aktion werden Sie nur benötigen, wenn Sie den SSL-Zugriff auf einen Server erlauben wollen, der nicht Port 445 benutzt. Im Normalfall brauchen Sie die Aktion dagegen nicht.

session-cookies-only manipuliert Cookies so, daß sie vom Browser nach dem Ende der Sitzung nicht gespeichert werden.

set-image-blocker bestimmt, wodurch eine gefilterte Graphik ersetzt wird. Als Parameter erhält die Aktion entweder `pattern`, `blank` oder eine URL. Voreingestellt ist `pattern`, wodurch statt der Graphik ein Schachbrettmuster angezeigt wird. Mit `blank` wird statt dessen eine transparente Graphik eingesetzt, wodurch nicht mehr erkennbar ist, daß überhaupt eine Graphik gefiltert wurde. Ist schließlich eine URL übergeben worden, so wird jede gefilterte Graphik durch die Graphik ersetzt, auf die die URL verweist.

Um die Konfiguration zu vereinfachen, definiert *user.action* zuerst einmal mehrere Aliase, wobei der Alias `allow-popups` in der mir vorliegenden Version nicht funktioniert. Statt

```
allow-popups       = -filter{popups} -kill-popups
```

müßte es wohl heißen:

```
allow-popups       = -filter{unsolicitated-popups} -kill-popups \
                     -filter{all-popups}
```

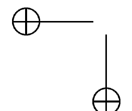
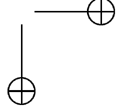
Dies bewirkt, daß keine Pop-up-Fenster mehr gefiltert werden.

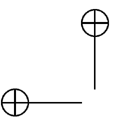
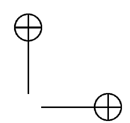
Daneben werden noch die folgenden Aliase definiert:

allow-all-cookies Cookies werden in keiner Weise gefiltert.

-crunch-all-cookies ähnelt `allow-all-cookies`, hat aber keine Auswirkungen auf den Filter `content-cookies`, der Cookies unterdrückt, die nicht vom Webserver gesetzt wurden, sondern in der Webseite selbst enthalten sind (z. B. in Form spezieller Tags oder als JavaScript-Befehle). Dieser Filter ist standardmäßig aber nicht aktiviert. Er verhindert ebenfalls nicht, daß Cookies so manipuliert werden, daß der Browser sie zum Sitzungsende löscht. Dieses Feature ist standardmäßig aktiviert, sollte aber im Normalfall keine Probleme machen.

+crunch-all-cookies unterdrückt alle Cookies.

+block-as-image Bedeutet, daß die spezifizierten URLs zu Graphiken gehören, die unterdrückt werden sollen. In den Standardeinstellungen bedeutet dies, daß die Graphik durch ein Schachbrettmuster ersetzt wird.

Diese Aktion sollte man verwenden, wenn man sicher ist, daß die zugehörige URL eine Graphik darstellt. Will man dagegen ganze Bereiche eines Servers filtern, sollte man lieber die Standardaktion `+block` benutzen. Diese versucht herauszufinden, ob es eine Graphik ist, und zeigt abhängig vom Ergebnis entweder das Schachbrettmuster oder einen Text an.

-block-as-image Ist eine andere Bezeichnung für `-block`. Diese Aktion hebt eine vorhergehende Sperrung mit `+block` oder `+block-as-image` auf.

fragile schaltet praktisch alle Filter für die angegebene URL ab. Dies ist sinnvoll für Seiten, die besonders empfindlich auf die Filterung reagieren.

shop steht für `-crunch-all-cookies` und `allow-popups`.

Neben Aktionen müssen wir aber auch URLs angeben, auf welche die Aktionen angewendet werden. Neben dem Muster `/`, das auf alle URLs paßt, kann man Muster in der folgenden Form angeben:

```
[<Rechner>][/<Pfad>]
```

<*Rechner*> kann eine vollständige Rechneradresse sein, es sind aber auch allgemeinere Formen möglich. Man kann Wildcards benutzen:

******* entspricht beliebig vielen Zeichen, einschließlich keinem Zeichen.

? entspricht genau einem Zeichen.

[1-9az] entspricht einer Ziffer zwischen 1 und 9, a oder z.

Man kann auch Adreßangaben mit einem Punkt beginnen oder abschließen. In diesem Fall dürfen auf der anderen Seite beliebige Zeichen stehen:

.example.com Es werden alle Rechner gefunden, die auf `.example.com` enden.

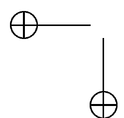
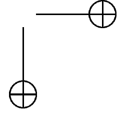
www. Dies beschreibt Rechnernamen, die mit `www.` beginnen.

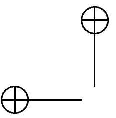
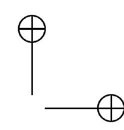
.example. Dieses Muster paßt auf alle Rechner, deren Name `.example.` enthält.

Für <*Pfad*> können beliebige Perl-kompatible reguläre Ausdrücke verwendet werden. Diese entsprechen in etwa den in Kapitel 16, Unterabschnitt *Künstliche Ignoranz*, ab Seite 508 beschriebenen erweiterten regulären Ausdrücken. Eine tiefergehende Einführung finden Sie unter *http://www.pcre.org/man.txt*.

Ich würde hier gerne ein paar Beispiele anführen, aber der Privoxy ist relativ gründlich. Von den Seiten, die ich regelmäßig frequentiere, fiel mir nur beim Betrachten der Dilbert-Comics auf *unitedmedia.com* Werbung auf, die der Proxy nicht filterte, darunter

```
http://c4.maxserving.com/iserver/site=5311/area=homepage/aamsz=banner/PageID=1234567
http://ak.maxserving.com/images/port/636/Bored_int_468x60.gif
```

In beiden Fällen stammten die Graphiken aus der Domäne *maxserving.com*. Auch wenn sie von verschiedenen Servern stammen, die *c4* und *ak* heißen, und obwohl die Pfade verschieden sind, kann man sie mit einer Regel im Abschnitt `{ +block-as-image }` ganz einfach filtern:

```
{ +block-as-image }
.maxserving.com
```

Glücklicherweise liegen sowohl die eigentliche Webseite wie auch die enthaltenen Comics in der Domäne *unitedmedia.com*, wodurch diese Regel tatsächlich nur die Werbebanner filtert.

Davon abgesehen wurde ich auf allen Seiten, die ich besuchte, von Schachbrettmustern begrüßt, die anzeigten, wo Werbebanner gefiltert worden waren.

Wenn Ihnen die Schachbrettmuster nicht gefallen, können Sie sie ganz einfach loswerden. Sie verlieren dann aber den Überblick darüber, wo Banner gefiltert wurden. Dies kann dann problematisch werden, wenn eine Graphik versehentlich gefiltert wurde. Ist es Ihnen dieses Risiko wert, so brauchen Sie nur die folgenden Zeilen in die Datei *user.action* einzutragen:

```
{ +set-image-blocker{blank} }
/
```

Automatischer Start beim Booten

Unter SuSE sollten Sie nun noch dafür sorgen, daß der Dienst beim Booten automatisch gestartet wird:

```
# insserv privoxy
```

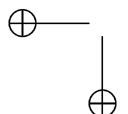
Unter Debian ist dies nicht nötig. Die entsprechenden Runlevel-Links wurden schon bei der Installation angelegt.

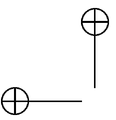
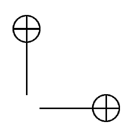
ftp-proxy aus der SuSE Proxy-Suite

Die SuSE Proxy-Suite ist ein Projekt von SuSE, das es sich zur Aufgabe macht, einen Satz sicherer Proxies für die Verwendung auf Firewalls zu entwickeln. Derzeit besteht die Suite allerdings ausschließlich aus einem FTP-Proxy.

Obwohl er von SuSE entwickelt wurde, ist der Proxy frei verfügbar und ist außer in SuSE-Linux z. B. auch in Debian 3.1 enthalten.

Der Proxy kann auf zwei Arten verwendet werden. Als normaler FTP-Proxy erlaubt er Benutzern des lokalen Netzes, auf FTP-Server im Internet zuzugreifen. Hierbei können im Gegensatz zu einer reinen Paketfilterung nicht nur die Adressen der beteiligten Rechner, sondern auch die verwendeten Befehle protokolliert werden. Ist der Proxy mit Unter-



stützung für `libwrap` kompiliert worden[6], so kann man wie beim `tcpd`[7] mit den Dateien */etc/hosts.allow* und */etc/hosts.deny* festlegen, von welchen Rechnern aus der Proxy benutzt werden darf und von welchen nicht.

Eine besonders interessante Einsatzmöglichkeit dieses Proxys ist darüber hinaus, daß er auch als transparenter Proxy genutzt werden kann.

Eine zweite Möglichkeit besteht darin, den `ftp-proxy` als Reverse Proxy einzusetzen. Dies ist dann sinnvoll, wenn man einen eigenen FTP-Server betreibt. Die Klienten im Internet verbinden sich hierbei nicht direkt mit dem FTP-Server, sondern mit dem Proxy, der alle Befehle entgegennimmt und nur diejenigen weitergibt, die sinnvoll und notwendig sind. Alle anderen werden vom Proxy mit einer Fehlermeldung verweigert. Auf diese Weise können Angriffe auf schlecht konfigurierte FTP-Server erschwert werden.

Schließlich ist er auch noch ausdrücklich dafür vorgesehen, in einer `chroot`-Umgebung eingesetzt zu werden. Das bedeutet, daß ein Angreifer, der den Proxy dazu bringen kann, beliebige Befehle auszuführen, in einem festgelegten Verzeichnis wie in einem Käfig gefangen ist und keine Dateien und Verzeichnisse außerhalb sehen, geschweige denn auf diese zugreifen kann.

Anlegen eines eigenen Benutzers

Bevor wir damit beginnen, den eigentlichen Proxy zu konfigurieren, sollten wir als erstes einen eigenen Benutzer und eine eigene Gruppe für den Dienst anlegen. Dadurch können wir später festlegen, welche Rechte der Dienst hat. Würde der Dienst z. B. mit den Rechten von `root` laufen, so könnte ein Fehler in der Programmierung von `ftp-proxy` dazu führen, daß ein Angreifer vollen Zugriff auf alle Ressourcen des Systems hätte.

Als erstes benötigen wir ein Verzeichnis, das wir als Heimatverzeichnis für unseren Proxy-User eintragen können. Wir wollen dazu */var/lib/ftp-proxy/rundir* verwenden. Falls es noch nicht existiert, können wir es mit dem folgenden Befehl anlegen:

```
# mkdir -p -m0755 /var/lib/ftp-proxy/rundir
```

Beachten Sie hierbei, daß das Verzeichnis `root` gehört und niemand sonst Schreibrecht darauf hat. Dies wird auch unseren zukünftigen Proxy-User einschließen.

Legen wir nun eine Gruppe an, die wir als Standardgruppe für unseren Benutzer verwenden werden:

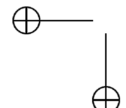
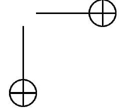
```
# groupadd ftp-proxy
```

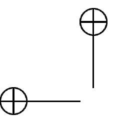
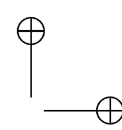
Schließlich muß noch der eigentliche Systembenutzer angelegt werden:

---

6 Wenn dies der Fall ist, zeigt `ldd` *<Pfad>*`/ftp-proxy` unter anderem die Bibliothek `libwrap.so.`*<Version>* an.

7 Vgl. Kapitel 9, Abschnitt *Der inetd*, ab Seite 182


```
# useradd -c ßystem user for ftp-proxy" \
> -d /var/lib/ftp-proxy/rundir \
> -g ftp-proxy \
> -G '' \
> -s /bin/false \
> ftp-proxy
```

Einrichtung einer chroot-Umgebung

Wir wollen den Systembenutzer aber nicht nur in seinen Rechten beschränken, sondern noch einen Schritt weiter gehen. Wir werden dem Proxy einen Käfig bauen, aus dem er nicht ausbrechen kann. Hierzu müssen wir ein Verzeichnis schaffen, in dem die Verzeichnisstruktur des Systems in dem Maße nachgebildet wird, daß der Proxy dort alle Dateien vorfindet, die er für seine Arbeit benötigt. Wir konfigurieren ihn dann so, daß er einen `chroot`-Systemaufruf durchführt, wodurch dieses Verzeichnis für ihn zum Wurzelverzeichnis wird. Verzeichnisse und Dateien außerhalb seines Käfigs nimmt er nicht mehr wahr. Sie existieren für ihn nicht mehr.

Unter SuSE existiert ein solches Verzeichnis bereits, und alle nötigen Dateien werden beim Start des Dienstes hineinkopiert. Allerdings hat es unter SuSE 9.3 einen Bug. Es sucht nach Dateien */lib/libdb**, die nicht leer sein dürfen und ausführbar sein müssen. Wird keine solche Datei gefunden, so beendet sich das Runlevel-Skript ohne weitere Ausgaben. Auch das normale `OK` oder `Failed` wird nicht angezeigt. Lediglich im Systemprotokoll findet sich eine Meldung, die auf das Problem hinweist.

Glücklicherweise gibt es eine einfache Lösung. Wir legen einfach eine harmlose Datei an:

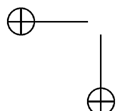
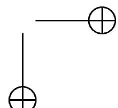
```
# echo -e '\000\000' > /lib/libdb.dummy
# chmod a+x /lib/libdb.dummy
```

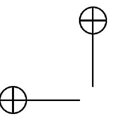
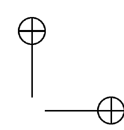
Nun läuft das Runlevel-Skript sauber durch und erledigt die im folgenden beschriebenen Schritte automatisch, so daß wir zum nächsten Abschnitt vorblättern können.

Unter Debian müssen wir */var/lib/ftp-proxy/rundir* dagegen selber bevölkern. Der korrekte Aufbau eines solchen Verzeichnisses ist normalerweise nicht ganz einfach, da man alle Ressourcen kennen muß, die der einzusperrende Dienst benötigt. Dazu gehören z. B. das Programm selbst, seine Konfigurationsdateien, dynamische Bibliotheken sowie Devices.

Glücklicherweise liegt den Quellen zum `ftp-proxy` ein Skript bei, das einem diese Arbeit abnimmt[8]. Allerdings läuft es nur unter SuSE-Linux. Statt es soweit zu patchen, daß es auch unter anderen Distributionen läuft, habe ich es mir genauer angesehen und die einzelnen Schritte manuell nachvollzogen. Die folgende Beschreibung ist das Ergebnis dieser Betrachtung.

8 Sie finden die aktuellen Quellen unter
ftp://ftp.suse.com/pub/projects/proxy-suite/src
SuSE hat auch eine Homepage zu der Software:
http://www.suse.de/en/whitepapers/proxy_suite/

Als erstes benötigen wir eine Verzeichnishierarchie mit allen relevanten Verzeichnissen. Die Verzeichnisse sollten `root` gehören und sonst für niemanden schreibbar sein. Wir legen diese Verzeichnisse im Heimatverzeichnis des Dienstes, also */var/lib/ftp-proxy/rundir* an:

```
# cd /var/lib/ftp-proxy/rundir
# mkdir -p -m0755 dev
# mkdir -p -m0755 etc/proxy-suite
# mkdir -p -m0755 lib
# mkdir -p -m0755 usr/sbin
# mkdir -p -m0755 usr/lib
# mkdir -p -m0755 var/lib/ftp-proxy/rundir
```

Als einzige Devices benötigt der Dienst das Null-Device und *ipnat*:

```
# mknod dev/null c 1 3
# chmod 0666 dev/null
# mknod dev/ipnat c 95 1
# chmod 0440 dev/ipnat
# chown ftp-proxy:ftp-proxy dev/ipnat
```

Auch das eigentliche Programm `ftp-proxy` wird benötigt:

```
# cp -fp /usr/sbin/ftp-proxy usr/sbin/ftp-proxy
```

Die meisten Programme sind dynamisch gelinkt. D. h., viele Funktionen sind nicht im eigentlichen Programm enthalten, sondern in Bibliotheken, die erst zur Laufzeit vom Programm geöffnet werden.

Diese Bibliotheken müssen wir ebenfalls in unseren Käfig kopieren. Welche dies sind, kann uns zumindest teilweise der Befehl `ldd` verraten:

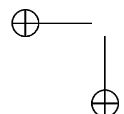
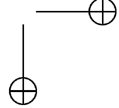
```
# ldd /usr/sbin/ftp-proxy
        libwrap.so.0 => /lib/libwrap.so.0 (0x4001d000)
        libnsl.so.1 => /lib/libnsl.so.1 (0x40025000)
        libc.so.6 => /lib/libc.so.6 (0x4003a000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

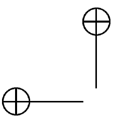
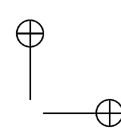
Kopieren wir die gefundenen Bibliotheken[9] nun herüber:

```
# cp -fp /lib/libwrap.so.0 lib/
# cp -fp /lib/libnsl.so.1 lib/
# cp -fp /lib/libc.so.6 lib/
# cp -fp /lib/ld-linux.so.2 lib/
```

Leider erzählt uns `ldd` nur die halbe Geschichte. Es existieren weitere Bibliotheken, die von den Bibliotheken der Glibc (d. h. *libc.so* und *libnsl.so*) nur dann eingebunden werden, wenn die aktuelle Konfiguration des Systems es erfordert. Da wir nicht genau wissen, welche Bibliotheken dies im konkreten Einzelfall sind, wollen wir hier großzügig alle in Frage kommenden Bibliotheken kopieren:

9 Genaugenommen ist `ld-linux.so.2` keine Bibliothek, sondern ein Hilfsprogramm, das zum eigentlichen Einbinden der Bibliotheken benötigt wird.


```
# cp -fp /lib/libnss* lib/
# cp -fp /lib/libnsl* lib/
# cp -fp /lib/libresolv* lib/
# cp -fp /lib/libdb* lib/
```

Beim Einbinden von Bibliotheken wird eine Datei */etc/ld.so.cache* benötigt. Diese erzeugen wir mit dem Befehl `ldconfig`:

```
# ldconfig -X -r /var/lib/ftp-proxy/rundir
ldconfig: /lib/libdb2.so.2 is not a symbolic link

ldconfig: /lib/libdb.so.2 is not a symbolic link
```

Mit der Option `-r` geben wir dabei vor, daß der Befehl alle Dateien relativ zu unserem neuen Wurzelverzeichnis öffnen soll. Dadurch werden die Bibliotheken unter */var/lib/ftp-proxy/rundir/lib* gesucht und in der Datei */var/lib/ftp-proxy/rundir/etc/ld.so.cache* registriert.

Als nächstes konfigurieren wir die Zeitzone unseres Systems. Dies geschieht normalerweise durch einen symbolischen Link */etc/localtime*, der auf eine Datei verweist, welche die eigentlichen Konfigurationsinformationen für unsere Zeitzone enthält:

```
# ls -l /etc/localtime
lrwxrwxrwx    1 root     root          33 Feb 17 21:26 /etc/localtime
-> /usr/share/zoneinfo/Europe/Berlin
```

Diese Datei kopieren wir in unseren Käfig:

```
# mkdir -p -m0755 usr/share/zoneinfo/Europe
# cp -fp  /usr/share/zoneinfo/Europe/Berlin usr/share/zoneinfo/Europe
```

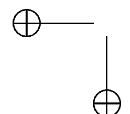
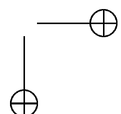
Damit `ftp-proxy` sie auch findet, müssen wir nun auch noch einen entsprechenden Link anlegen:

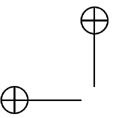
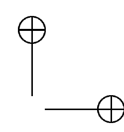
```
# cd etc
# ln -s ../usr/share/zoneinfo/Europe/Berlin localtime
# cd ..
```

Auch einige grundlegende Konfigurationsdateien werden im Käfig benötigt:

```
# for c in "/etc/host.conf
> /etc/resolv.conf
> /etc/hosts
> /etc/services
> /etc/protocols
> /etc/nsswitch.conf"
> do
> cp -fp $c etc/
> done
```

An dieser Stelle kopiert das SuSE-Skript noch mehr Dateien, aber damit schießt es meiner Meinung nach über das Ziel hinaus, da diese Dateien Daten enthalten, die für die Arbeit des Proxys unnötig sind. Die Grundannahme beim Einrichten eines solchen Käfigs ist nun aber, daß ein Angreifer es schaffen könnte, tatsächlich bis in ihn vorzudringen. Aus

diesem Grund sollten sich dort so wenig zusätzliche Informationen befinden, wie dies möglich ist, ohne die Funktion des Proxys zu behindern.

Ein Beispiel hierfür sind die Dateien */etc/passwd* und */etc/group*. Diese werden vom Skript ohne weitere Anpassungen in den Käfig kopiert. Damit kann ein eventueller Angreifer nachlesen, welche Benutzerkonten und Gruppen auf dem Rechner existieren.

Für den Proxy sind aber maximal die Benutzer `root` und `ftp-proxy` bzw. die gleichnamigen Gruppen interessant. Alle anderen Benutzerkonten und Gruppen werden von ihm nicht benötigt. Wir wollen daher im folgenden im Käfig Dateien erzeugen, die auch nur diese Informationen enthalten:

```
# grep '^root' /etc/passwd > etc/passwd
# grep '^ftp-proxy' /etc/passwd >> etc/passwd
# chmod 644 etc/passwd
# grep '^root' /etc/group > etc/group
# grep '^ftp-proxy' /etc/group >> etc/group
# chmod 644 etc/group
```

Konfiguration des Systemprotokolldienstes

Als nächstes müssen wir sicherstellen, daß auch Meldungen in das Systemprotokoll geschrieben werden können. Normalerweise werden Ereignisse protokolliert, indem auf den Unix Domain Socket */dev/log* geschrieben wird.

Hierbei handelt es sich nicht um ein Device, das wir kopieren können, sondern um ein Objekt, das vom `syslogd` zur Laufzeit angelegt wird. Wir müssen daher den Systemprotokolldienst so konfigurieren, daß er zusätzlich zu */dev/log* auch */var/lib/ftp-proxy/rundir/dev/log* anlegt.

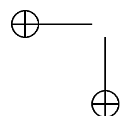
Hierbei müssen wir zwischen dem normalen `syslogd`, wie er standardmäßig unter Debian eingesetzt wird, und dem `syslog-ng` unterscheiden, den SuSE mittlerweile einsetzt.

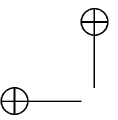
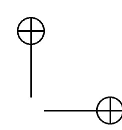
Benutzen wir den `syslog-ng`, so brauchen wir nur eine zusätzliche Quellenangabe in seine Konfigurationsdatei einzufügen:

```
source src {
[...]
        unix-dgram("/var/lib/ftp-proxy/rundir/dev/log");
[...]
};
```

Diese Datei finden wir unter */etc/syslog-ng/syslog-ng.conf* (siehe Kapitel 9, Unterabschnitt *syslog-ng*, ab Seite 208).

Beim `syslogd` müssen wir dafür sorgen, daß er beim Systemstart mit dem zusätzlichen Parameter `-a /var/lib/ftp-proxy/rundir/dev/log` aufgerufen wird. Wir suchen also als erstes das Runlevel-Skript, in dem der `syslogd` gestartet wird:




```
# find /etc/init.d -type f -exec grep -Hn syslogd \{\} \;
/etc/init.d/klogd:35:    # No syslogd?
/etc/init.d/sysklogd:6:pidfile=/var/run/syslogd.pid
/etc/init.d/sysklogd:7:binpath=/sbin/syslogd
/etc/init.d/sysklogd:46:    # No syslogd?
/etc/init.d/sysklogd:58:    echo -n "Starting system log daemon: syslogd"
/etc/init.d/sysklogd:64:    echo -n "Stopping system log daemon: syslogd"
/etc/init.d/sysklogd:72:    echo -n "Stopping system log daemon: syslogd"
/etc/init.d/sysklogd:76:    echo -n "Starting system log daemon: syslogd"
```

Gegebenenfalls müssen Sie hier das Verzeichnis, in dem gesucht wird, anpassen. Debian und SuSE 9.3 verwenden */etc/init.d*, aber andere Distributionen haben auch z. B. */sbin/init.d* oder */etc/rc.d/init.d* benutzt.

Unter Debian finden wir die Datei */etc/init.d/sysklogd*. Diese kennt eine Variable *SYSLOGD*, die zusätzliche Parameter enthalten kann, im Normalfall aber leer ist.

Hier brauchen wir nur unseren Parameter anzufügen:

```
SYSLOGD="-a /var/lib/ftp-proxy/rundir/dev/log"
```

Es macht dabei überhaupt nichts, wenn schon ein Parameter `-a ...` eingetragen ist. Wir können durchaus einen eigenen Wert anhängen:

```
SYSLOGD="-a <Anderer> -a /var/lib/ftp-proxy/rundir/dev/log"
```

SuSE ist in dieser Hinsicht komplizierter. Hier stehen solche Konfigurationsvariablen nicht im eigentlichen Runlevel-Skript, sondern in einer eigenen Konfigurationsdatei. Früher war dies */etc/rc.config*, vor einiger Zeit wurde diese zentrale Datei aber in eine ganze Reihe kleinerer Dateien aufgeteilt, die spezifisch für jeweils eine Applikation sind.

Der `syslogd` wird in SuSE 9.3 in der Datei */etc/sysconfig/syslog* konfiguriert. Wir haben hier zwei Möglichkeiten, zusätzliche Unix Domain Sockets anzugeben.

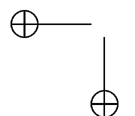
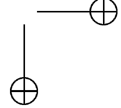
Zum einen kennt das Skript eine allgemeine Variable, in der ganz allgemein zusätzliche Parameter übergeben werden können:

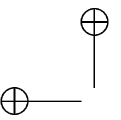
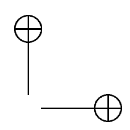
```
SYSLOGD_PARAMS="-a /var/lib/ftp-proxy/rundir/dev/log"
```

Alternativ können auch eigene Variablen angegeben werden, die jeweils einen zusätzlichen Unix Domain Socket angeben. Der Name einer solchen Variablen muß mit `SYSLOGD_ADDITIONAL_SOCKET` anfangen. Die Variable enthält jeweils den Namen des Socket ohne das `-a` (also z.B: `SYSLOGD_ADDITIONAL_SOCKET_FTPPROXY=/var/lib/ftp-proxy/rundir/dev/log`).

TCPWrapper-Konfiguration

Der Proxy kann so konfiguriert werden, daß er die Dateien */etc/hosts.allow* und */etc/hosts.deny* konsultiert, um zu entscheiden, ob ein bestimmter Rechner überhaupt eine Verbindung aufbauen darf.

Diese Dateien haben wir in */etc/* bereits überprüft (siehe Kapitel 9, Abschnitt *Der inetd*, ab Seite 182). Im Käfig müssen sie aber erst noch angelegt werden. Dabei sollten wir nicht einfach die Dateien aus */etc/* kopieren, da sie unter Umständen Angaben zu Diensten enthalten, die für den Käfig nicht relevant sind.

Stattdessen erzeugen wir Minimaldateien, die nur die absolut unerläßlichen Angaben enthalten:

```
# cd /var/lib/ftp-proxy/rundir
# touch etc/hosts.allow
# chmod 644 etc/hosts.allow
# echo ÄLL : ALL" > etc/hosts.deny
# chmod 644 etc/hosts.deny
```

Nun müssen wir nur noch eine Zeile in beide Versionen der Datei */etc/hosts.allow* eintragen, die die gewünschten Zugriffe erlaubt. In meinen Experimenten schienen zwar nur die Dateien im Käfig verwendet zu werden, für dieses durchaus logische Verhalten habe ich aber keine Dokumentation gefunden. Sicherheitshalber sollte sich die Konfiguration im Hinblick auf den `ftp-proxy` in */etc/* und */var/lib/ftp-proxy/rundir/etc/* nicht unterscheiden.

Wenn der Proxy unseren Benutzern den Zugang auf Server im Internet erlauben soll, dann ist die folgende Zeile in *hosts.allow* angebracht:

```
ftp-proxy : 127.0.0.1, 192.168.20.0/255.255.255.0
```

Beim Einsatz als Reverse Proxy existiert dagegen normalerweise kein Grund, jemanden von der Benutzung des Proxys auszuschließen.

```
ftp-proxy : ALL
```

Soll aber im Einzelfall doch einmal ein bestimmter Host von der Benutzung ausgeschlossen werden, so kann das durch folgende Änderung geschehen:

```
ftp-proxy : ALL EXCEPT 172.16.51.3
```

Damit darf 172.16.51.3 nicht auf den Proxy zugreifen.

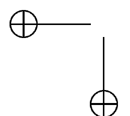
Grundkonfiguration des ftp-proxy

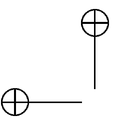
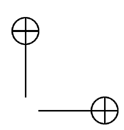
Nachdem wir das System vorbereitet haben, fehlt noch die Konfiguration des Programms selbst. Diese geschieht in der Datei */etc/proxy-suite/ftp-proxy.conf*.

Die Datei ist in Abschnitte eingeteilt. Der wichtigste Abschnitt ist dabei `-Global-`, der am Anfang der Datei steht:

```
[-Global-]
```

Weitere Abschnitte könnten benutzerspezifische Optionen enthalten. Wir werden aber hier darauf verzichten.



Die folgenden Optionen legen fest, aus welchem Bereich `ftp-proxy` den Quellport wählt, wenn er bei aktivem FTP eine Datenverbindung zum Klienten im lokalen Netz aufbaut. Gibt man hier nichts an, so verwendet er Port 20, falls er mit `root`-Rechten läuft.

Dies ist hier allerdings nicht der Fall, weshalb er einen beliebigen Port oberhalb von 1024 verwenden würde. Das würde es uns aber nahezu unmöglich machen, sinnvolle Firewallregeln zu definieren. Daher legen die folgenden Einträge den Port auf 2020 fest. Diese Nummer habe ich willkürlich gewählt. Sie können durchaus eine beliebige andere verwenden, solange sie deutlich größer als 1024 ist.

```
ActiveMinDataPort  2020
ActiveMaxDataPort  2020
```

Die folgende Option erlaubt es, den Zielrechner als Teil des Anmeldenamens in der folgenden Form anzugeben:

<Kennung>@<Zielsystem>


Diese Einstellung ist nur sinnvoll, wenn `ftp-proxy` nicht als reverse Proxy eingesetzt werden soll. Andernfalls sollten Sie unbedingt darauf verzichten, da sonst beliebige Benutzer die Möglichkeit haben, anonymisiert auf beliebige Rechner im Internet zuzugreifen. Diese Zugriffe würden aus Sicht des Zielrechners von unserer Firewall kommen. Wir würden also damit riskieren, für Angriffe Dritter auf FTP-Server verantwortlich gemacht zu werden.

Für normale Proxies sollten wir die folgende Zeile eintragen:

```
AllowMagicUser     yes
```

Für Reverse Proxies dagegen:

```
AllowMagicUser     no
```

Mit der nächsten Option können Sie den `ftp-proxy` als transparenten Proxy einsetzen. Dazu müssen wir die Paketfilterung so konfigurieren, daß Zugriffe auf Server im Internet auf den Proxy umgeleitet werden. Dieser macht dann eine Verbindung zum gewünschten Server auf, ohne daß der Benutzer diesen noch einmal explizit angeben muß.

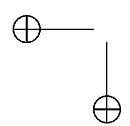
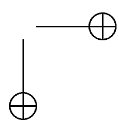

Auch diese Einstellung ist nur für normale Proxies, nicht aber für den Einsatz als Reverse Proxy sinnvoll:

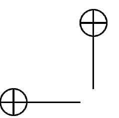
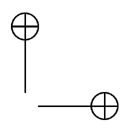
```
AllowTransProxy    yes
```

Wir können dem Proxy eine Zieladresse vorgeben, die er immer dann benutzt, wenn der Benutzer es versäumt, einen Zielrechner anzugeben:

```
DestinationAddress ftp.de.kernel.org
```

Diese Einstellung ist auch der Schlüssel zum Einsatz als Reverse Proxy. Dabei wollen wir ja, daß ein Zugriff auf den Proxy an unseren Server in der DMZ weitergeleitet wird. Hat

dieser also die Adresse 10.0.0.6, so brauchen wir den folgenden Eintrag in der Konfigurationsdatei:

```
DestinationAddress 10.0.0.6
```

Wir wollen nicht, daß unser Proxy mit `root`-Rechten läuft. Die folgende Option gibt an, mit welchen Gruppenrechten er statt dessen laufen soll:

```
Group              ftp-proxy
```

Wenn wir einen normalen Proxy betreiben, wollen wir nicht, daß er aus dem Internet angesprochen wird. Eine Maßnahme, um dies zu erreichen, besteht darin, explizit vorzugeben, daß nur Pakete angenommen werden sollen, die an die interne Adresse der Firewall gerichtet sind.

Für einen Reverse Proxy sollte die folgende Zeile dagegen nicht eingetragen werden:

```
Listen             192.168.20.15
```

Auch die Protokollierung der Zugriffe im Systemprotokoll ist eine definierte Aufgabe unseres Proxys. Die folgende Zeile weist ihn daher an, mit Hilfe des `syslog` zu protokollieren und dabei als Quelle `daemon` anzugeben. Diese ist für Systemdienste gedacht:

```
LogDestination     daemon
```

Wir haben ja schon alles vorbereitet, damit der Dienst in einem `chroot`-Käfig laufen kann. Die folgende Zeile weist ihn an, es auch zu tun:

```
ServerRoot         /var/lib/ftp-proxy/rundir
```

Grundsätzlich kann der `ftp-proxy` als eigenständiger Dienst laufen oder vom `inetd` aufgerufen werden. Wir wollen ihn als eigenständigen Server betreiben:

```
ServerType         standalone
```

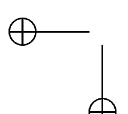
Wurde der Proxy mit der Option `-with-libwrap` kompiliert, dann kann man mit den folgenden Optionen einstellen, daß bei jedem Verbindungsaufbau eines Klienten erst einmal */etc/hosts.allow* und */etc/hosts.deny* konsultiert werden, um zu entscheiden, ob dieser Rechner auf den Proxy überhaupt zugreifen darf:

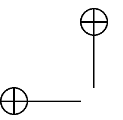
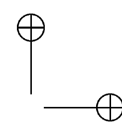
```
TCPWrapper         yes
TCPWrapperName     ftp-proxy
```

Nachdem wir schon festgelegt haben, mit welchen Gruppenrechten `ftp-proxy` laufen soll, legt die folgende Einstellung den zu verwendenden Benutzer fest:

```
User               ftp-proxy
```

Mit dem `ftp-proxy` können wir auch festlegen, welche FTP-Befehle überhaupt verwendet werden dürfen. Diese Möglichkeit ist grundsätzlich ein sehr mächtiges Werkzeug,



das allerdings nicht einfach zu konfigurieren ist. Der Proxy kennt 51 Befehle[10], die wir erlauben oder verbieten können.

Die meisten davon habe ich in den RFCs 959, 1579, 1639, 2228, 2389, 2428 und 2640 gefunden, wobei RFC 959 definitiv am wichtigsten ist. Die anderen RFCs sind Erweiterungen, die oft nur einzelne Befehle definieren. Einige Befehle habe ich aber immer noch nicht gefunden, oder ihre Funktion wurde in einem Nebensatz vage erwähnt, aber nicht wirklich definiert.

Für einen Reverse Proxy würde ich die folgenden Befehle erlauben. Sie gestatten es, Dateien herunterzuladen, nicht aber eigene Dateien auf dem Server abzulegen:

```
ValidCommands       USER, PASS, PWD, CWD, \
                    PORT, PASV, RETR, TYPE, \
                    REST, ABOR, LIST, NLST, \
                    STAT, QUIT
```

Dies sollte für den Betrieb eines Reverse Proxys ausreichen, wenn der FTP-Server nur dazu dient, Informationen bereitzustellen, nicht aber als Upload-Server z. B. für Webseiten dient.

Für einen normalen Proxy werden Sie wahrscheinlich mehr Befehle erlauben müssen. Vermutlich werden die Benutzer irgendwann auch Dateien senden und nicht nur empfangen wollen. In diesem Fall werden Sie um das Studium von RFC 959 und eigene Experimente mit diversen FTP-Klienten nicht herumkommen. Hilfreich dürfte dabei vor allem die Tatsache sein, daß jeder Befehl im Systemprotokoll vermerkt wird.

Fehlt die Option komplett, so sind alle Befehle erlaubt, die der Proxy kennt, darunter auch so gefährliche wie `SITE`, die den FTP-Server anweisen, lokale Kommandos auszuführen.

Haben wir die Datei zu unserer Zufriedenheit konfiguriert, so müssen wir sie nur noch in unseren `chroot`-Käfig kopieren:

```
# cd /var/lib/ftp-proxy/rundir
# cp /etc/proxy-suite/ftp-proxy.conf etc/proxy-suite/
```

Falls Sie den in Debian enthaltenen `ftp-proxy` verwenden, denken Sie bitte daran, in den obigen Befehlen *etc/proxy-suite/* jeweils durch *etc/* zu ersetzen.

Tests

Nun wollen wir wissen, ob unser Proxy auch tatsächlich das tut, was wir von ihm erwarten.

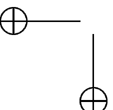
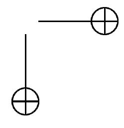
Dazu müssen wir ihn als erstes starten:

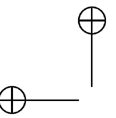
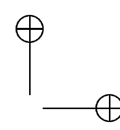
```
# /etc/init.d/ftp-proxy start
```

Passen Sie dabei bitte den Pfad und den Skriptnamen entsprechend Ihrer Installation an.



10 Details unter `man ftp-proxy.conf`

Nun verbinden wir uns zum ersten Mal mit dem Proxy:

```
# ftp localhost
Connected to gonzales.
220 gonzales FTP server (Version 1.9.2.2 - 2004/03/30 10:42:46) ready.
Name (localhost:perdita):
```

Dies ist die Standardmeldung des Proxys. Sie kann umkonfiguriert werden, wenn man z. B. nicht will, daß einfach zu erkennen ist, daß ein Reverse Proxy eingesetzt wird.

Geben wir als Benutzernamen `ftp` oder `anonymous` ein, so sehen wir, ob unser Standardziel bzw. der eigene FTP-Server korrekt konfiguriert ist. Erscheint die folgende Zeile, so fehlt wahrscheinlich die Zeile `DestinationAddress`:

```
Name (localhost:perdita): ftp
501 Unknown destination address.
Login failed.
ftp>
```

Erscheint dagegen eine Aufforderung, Ihr Paßwort einzugeben, so wurden Sie mit dem Zielrechner verbunden. Testen Sie nun, ob Sie richtig auf den Server zugreifen können, indem Sie in verschiedene Verzeichnisse wechseln, sich die Verzeichnisinhalte anzeigen lassen und Dateien herunterladen. Falls Ihr FTP-Klient dies unterstützt, wechseln Sie zwischen Aktiv- und Passiv-Modus (z. B. mit dem Befehl `passive`), und geben Sie mit den Befehlen `ascii` und `binary` die Übertragung jeweils als Text- und Binärdatei vor.

Als nächstes sollten Sie den Zielrechner explizit vorgeben:

```
# ftp localhost
Connected to gonzales.
220 gonzales FTP server (Version 1.9.2.2 - 2004/03/30 10:42:46) ready.
Name (localhost:perdita): ftp@ftp.de.kernel.org
331 Please specify the password.
Password:
```

Dies sollte auf einem normalen Proxy funktionieren, bei einem Reverse Proxy aber keinesfalls.

Setzen Sie die Unterstützung für TCPWrapper ein, so können Sie testen, ob die Filterlisten auch tatsächlich angewendet werden, indem Sie ein Dummy-Interface auf eine Adresse konfigurieren, die keinen Zugriff haben sollte:

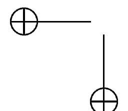
```
# ifconfig dummy0 172.16.23.23
```

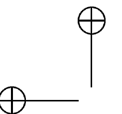
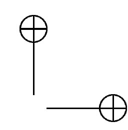
Nun können wir Netcat diese Adresse für eine Verbindung zum Proxy nutzen lassen:

```
# nc -s 172.16.23.23 localhost ftp
```

Nach einem Augenblick bricht der Befehl ohne eine Ausgabe ab, und im Systemprotokoll finden wir die Zeile

```
Apr 25 20:13:06 gonzales ftp-proxy[7572]: USER-ERR ftp-proxy reject:
'172.16.23.23' (Wrap)
```

Schließlich sollten wir noch Tests mit verschiedenen Klienten von einem Rechner im lokalen Netz durchführen. Insbesondere wenn wir die Anzahl der erlaubten Kommandos eingeschränkt haben, sollten wir anhand typischer Abläufe überprüfen, ob unsere Anwender sinnvoll arbeiten können.

Haben wir transparentes Proxying eingestellt, so ist jetzt der Zeitpunkt, es zu testen. Dazu ist es unbedingt nötig, den Test von einem eigenen Rechner im lokalen Netz aus durchzuführen. Von der Firewall aus kann diese Funktionalität nicht sinnvoll getestet werden.

Automatischer Start des Dienstes

Unter Debian wird der Dienst beim Booten automatisch gestartet, falls eine Konfigurationszeile der Art

```
ServerType standalone
```

existiert.

Unter SuSE müssen Sie den Dienst explizit mit dem folgenden Befehl aktivieren:

```
# insserv ftp-proxy
```

Einrichten eines DNS-Servers

Das Einrichten eines DNS-Servers auf der Firewall erlaubt es, Anfragen zu cachen, wodurch wiederholte Anfragen nach derselben Adresse nicht jedesmal zum Öffnen einer Verbindung in das Internet führen. Darüber hinaus bietet ein Proxy die Möglichkeit, Anfragen zu protokollieren und zentral festzulegen, welcher übergeordnete Server von den Clients befragt wird. Auch wenn sich dieser z. B. durch einen Provider-Wechsel ändert, muß nur an einer Stelle eine Anpassung vorgenommen werden.

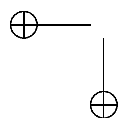
Die folgende Beschreibung geht von der Software Bind in der Version 9 aus. Bitte verwenden Sie keine älteren Versionen der Software, da diese teilweise gravierende Sicherheitslöcher enthalten.

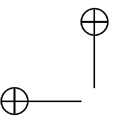
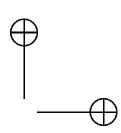
Falls Sie einen Kernel der Serie 2.2 einsetzen, sollten Sie mindestens 2.2.18 verwenden. Ältere Versionen unterstützen nicht die notwendige Funktionalität, um einen `chroot`-Käfig mit Bind zu benutzen. Außerdem sollten Sie grundsätzlich einen möglichst aktuellen Kernel verwenden.

Kernels späterer Serien (z. B. 2.4 und 2.6) sollten kein Problem darstellen.

Einrichtung eines chroot-Käfigs

Um die Gefahr, die sich aus der Kompromittierung des Dienstes ergibt, einzuschränken, sollten wir ihn in einem `chroot`-Käfig betreiben. Das bedeutet, wir starten ihn so, daß ein vorher festgelegtes Verzeichnis für den Prozeß zum Wurzelverzeichnis wird. Dateien au-



ßerhalb dieses Verzeichnisses sind damit für ihn nicht mehr sichtbar. Dieses Verzeichnis muß daher Kopien all jener Dateien und Verzeichnisse enthalten, die der Dienst für seine Arbeit benötigt.

Unter SuSE 9.3 existiert bereits ein solches Verzeichnis für den Bind, so daß die in diesem Abschnitt dargestellten Schritte entfallen.

Für Distributionen wie Debian 3.1, die von Haus aus den `chroot`-Betrieb nicht unterstützen, werden wir im folgenden sehen, wie ein solches Verzeichnis samt Inhalt erstellt wird.

Eine recht ausführliche Darstellung, wie das vor sich geht, findet man unter [8]. Die folgende Beschreibung entstand aus der Umsetzung dieser Darstellung auf dem Rechner des Autors.

Normalerweise würde man damit beginnen, erst einmal ein spezielles Benutzerkonto und eine eigene Gruppe für den Dienst zu erzeugen. Unter Debian 3.1 ist dies aber nicht mehr nötig. Hier existiert der Dienst standardmäßig mit den Rechten des Benutzers und der Gruppe `bind`.

Als Heimatverzeichnis für den Benutzer ist */var/cache/bind* angegeben und als Shell */bin/false*. Mitglied weiterer Gruppen außer `bind` ist der Benutzer `bind` nicht. Damit spricht nichts dagegen, dieses Benutzerkonto beizubehalten.

Wir können also gleich damit anfangen, daß wir uns für ein Verzeichnis entscheiden, in das wir den Dienst einsperren. Im folgenden soll dies */var/lib/named* sein.

Als erstes müssen wir das Verzeichnis und ein paar Unterverzeichnisse anlegen:

```
# mkdir -p -m0755 /var/lib/named
# cd /var/lib/named
# mkdir -p -m0755 etc/bind \
> dev \
> var/cache/bind
```

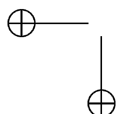
Einen Sonderfall stellt das Verzeichnis *var/run* dar. In dieses muß der Dienst schreiben können. Zu dem Zeitpunkt, wo er es versucht, hat er aber schon seine `root`-Rechte aufgegeben:

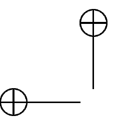
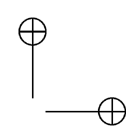
```
# mkdir -p -m0775 var/run
# chown root:bind var/run
```

Um den Dienst mit dem Hilfsprogramm `rndc` steuern zu können, wird eine Schlüsseldatei *rndc.key* benötigt. Damit diese auch im `chroot`-Käfig zur Verfügung steht, muß sie kopiert werden. Wo man sie findet, hängt von der verwendeten Distribution ab. Unter Debian wird */etc/bind/* verwendet. Die nötigen Befehle lauten hier:

```
# cp /etc/bind/rndc.conf etc/bind
# chown root:bind etc/bind/rndc.key
# chmod 440 etc/bind/rndc.key
```

Wir benötigen auch noch ein Verzeichnis, das wir als Verzeichnis für Zonendateien angeben werden. Wir werden hier */var/lib/named* verwenden. Da wir im `chroot`-Käfig die



normalen Verzeichnisstrukturen unterhalb des neuen Wurzelverzeichnisses nachbilden, würde daraus */var/lib/named/var/lib/named*. Wir können mit den folgenden Befehlen daraus aber auch einen symbolischen Link auf */var/lib/named* machen. Damit können die Dateien unter beiden Pfaden angesprochen werden:

```
# cd var/lib
# ln -s ../.. named
# cd ../..
```

Der Bind benötigt auch noch eigene Instanzen der Devices */dev/null* und */dev/random* in seinem `chroot`-Käfig:

```
# mknod dev/null c 1 3
# mknod dev/random c 1 8
# chmod 666 dev/null dev/random
```

Damit der Dienst auch weiß, in welcher Zeitzone er sich befindet, benötigt er die Datei */etc/localtime*. Üblicherweise ist dies allerdings ein nur symbolischer Link auf eine von mehreren Dateien, die jeweils verschiedene Zeitzonen definieren.

Im Fall von Debian 3.1 ergibt sich folgendes Bild:

```
# ls -l /etc/localtime
lrwxrwxrwx    1 root     root          33 Feb 17 21:26 /etc/localtime
-> /usr/share/zoneinfo/Europe/Berlin
```

Wir müssen also den entsprechenden Pfad auch in unserem `chroot`-Käfig erstellen und die eigentliche Zeitzonendatei hineinkopieren:

```
# mkdir -p -m0755 usr/share/zoneinfo/Europe
# cp -fp  /usr/share/zoneinfo/Europe/Berlin usr/share/zoneinfo/Europe
```

Nun müssen wir im Verzeichnis *etc* noch den Link anlegen.

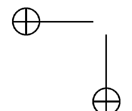
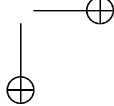
```
# cd etc
# ln -s ../usr/share/zoneinfo/Europe/Berlin localtime
# cd ..
```

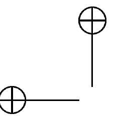
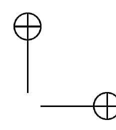
Sollte es sich in Ihrem Fall bei */etc/localtime* nicht um einen symbolischen Link handeln, so reicht es natürlich, die Datei einfach nach */var/lib/named/etc* zu kopieren.

Konfiguration des Systemprotokolldienstes

Schließlich müssen wir noch sicherstellen, daß der Bind Meldungen in das Systemprotokoll schreiben kann. Dazu muß er Kontakt zum Systemprotokolldienst aufnehmen. Normalerweise geschieht dies über den Unix Domain Socket */dev/log*. Dies ist ein spezielles Objekt, das vom Dienst zur Laufzeit angelegt wird.

Es liegt aber außerhalb unseres `chroot`-Käfigs. Wir müssen den Dienst daher dazu bringen, zusätzlich zu */dev/log* auch */var/lib/named/dev/log* zu benutzen.

syslog-ng

In Kapitel 9, Unterabschnitt *syslog-ng*, ab Seite 208 haben wir in der Konfigurationsdatei */etc/syslog-ng/syslog-ng.conf* ja bereits einen auskommentierten Eintrag für einen Socket im chroot-Verzeichnis des Bind vorgesehen. Nun brauchen wir nur noch das Kommentarzeichen # vor der entsprechenden Zeile zu entfernen:

```
source src {
[...]
        unix-dgram("/var/lib/named/dev/log");
[...]
};
```

Anschließend sollten wir den syslog-ng neu starten, um die Änderung wirksam werden zu lassen. Unter Debian verwenden wir dazu den Befehl:

```
# /etc/init.d/syslog-ng restart
```

Unter SuSE dagegen:

```
# /etc/init.d/syslog restart
```

syslogd

Verwenden wir den syslogd unter SuSE, so ist sichergestellt, daß ein Socket im chroot-Käfig vorhanden ist. Unter Debian müssen wir den syslogd erst entsprechend konfigurieren.

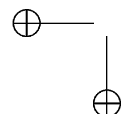
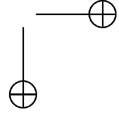
Dies geschieht, indem wir das Programm mit dem Parameter *-a /var/lib/named/dev/log* aufrufen. Dazu müssen wir das Runlevel-Skript finden, in dem der syslogd gestartet wird.

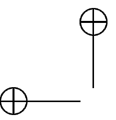
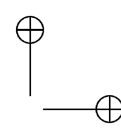
Bei Debian liegen Runlevel-Skripte unter */etc/init.d*, aber andere Distributionen haben auch z. B. */sbin/init.d* oder */etc/rc.d/init.d* benutzt:

```
# find /etc/init.d -type f -exec grep -Hn syslogd \{\} \;
/etc/init.d/klogd:35:    # No syslogd?
/etc/init.d/sysklogd:6:pidfile=/var/run/syslogd.pid
/etc/init.d/sysklogd:7:binpath=/sbin/syslogd
/etc/init.d/sysklogd:46:    # No syslogd?
/etc/init.d/sysklogd:58:    echo -n "Starting system log daemon: syslogd"
/etc/init.d/sysklogd:64:    echo -n "Stopping system log daemon: syslogd"
/etc/init.d/sysklogd:72:    echo -n "Stopping system log daemon: syslogd"
/etc/init.d/sysklogd:76:    echo -n "Starting system log daemon: syslogd"
```

Hier finden wir das Skript */etc/init.d/sysklogd*. Wenn wir es uns genauer ansehen, stellen wir fest, daß die Variable *SYSLOGD* es uns erlaubt, zusätzliche Parameter anzugeben, die dem Dienst beim Start mitgegeben werden:

```
SYSLOGD="-a /var/lib/named/dev/log"
```

Falls dort bereits ein entsprechender Parameter für einen anderen Dienst steht, so ist das kein Problem. Man kann auch mehrere Sockets für verschiedene Dienste angeben:

```
SYSLOGD="-a <Anderer> -a /var/lib/named/dev/log"
```

Grundkonfiguration

Wir wollen nun einen DNS-Server aufsetzen, der nur den Rechner *localhost.loopback* mit der Adresse *127.0.0.1* kennt und alle anderen Anfragen an übergeordnete DNS-Server eines Providers weiterleitet.

Der Eintrag weiterer Adreßbereiche auf dem DNS-Server ist kein Problem und kann später dazu genutzt werden, beispielsweise die Rechner des lokalen Netzes ebenfalls mitzuverwalten. Dazu müssen wir allerdings sicherstellen, daß externe Rechner keine Anfragen an unseren Server stellen dürfen.

Beginnen wir mit der Konfigurationsdatei des Servers. Würden wir den Dienst nicht in einem chroot-Käfig betreiben, wäre es die Datei */etc/named.conf* (Debian: */etc/bind/named.conf*).

In unserem Fall ist es etwas komplizierter. Grundsätzlich liest der Dienst die Datei im chroot-Verzeichnis. Allerdings kopiert das SuSE-Runlevel-Skript beim Start des Dienstes die Datei aus */etc* nach */var/lib/named/etc*. Unter SuSE 9.3 müssen wir also die Datei */etc/named.conf* ändern.

Debian 3.1 kopiert die Datei nicht um. Hier müssen wir daher die Datei im chroot-Käfig ändern. Der komplette Dateiname lautet also */var/lib/named/etc/bind/named.conf*. Am besten fangen wir dabei direkt an, indem wir eine neue Datei erstellen.

Zuerst definieren wir den Pfad, unter dem der Server seine Dateien findet. Wir haben uns ja bereits für */var/lib/named* entschieden:

```
#
# Globale Optionen
#

options {

        # Pfad für Dateien des Servers

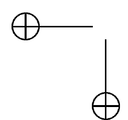
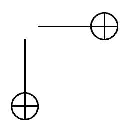
        directory "/var/lib/named";
```

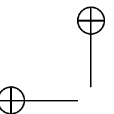
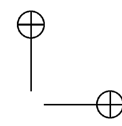
Als nächstes wollen wir einstellen, daß Anfragen nur an bestimmte DNS-Server weitergeleitet, nicht aber eigenständig durch Suche nach dem zuständigen DNS-Server beantwortet werden.

```
        # Antworten dürfen nur von den Forwardern erbeten werden.
        # Eigene Anfragen bei anderen Servern sind nicht zulässig.

        forward only;
```



```
# DNS-Server

forwarders{
       10.0.0.77;
};
```

Nun müssen wir sicherstellen, daß unser Server Anfragen nur der Rechner beantwortet, denen wir ausdrücklich vertrauen. Dies gilt insbesondere für Zone Transfers.

```
# Zone Transfers
allow-transfer {127.0.0.1; };

# Normale Anfragen
allow-query    {127.0.0.1; 192.168.20.0/24; };
```

Bevor wir nun zu den einzelnen Zonen kommen, könnten wir uns noch die Frage stellen, ob wir wollen, daß Klienten die Versionsnummer des Bind durch eine einfache DNS-Abfrage nach »version.bind« in der Klasse `chaos` abfragen dürfen. Wenn nicht, empfiehlt sich der Eintrag:

```
version "you surely must be joking";
```

Wichtig ist auch noch, den Bereich »options« sauber zu beenden:

```
};
```

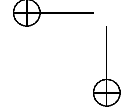
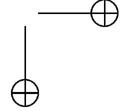
Schließlich können wir definieren, welche Zonen der Server kennen soll. Jede Zone sollte in zwei Formen definiert werden. Einmal als logischer Name »dummy.dom«, um diesen IP-Adressen der Art »1.2.3.x« zuweisen zu können, und einmal als IP-Adreßbereich in der Form »3.2.1.in-addr.arpa«, um auch aus den IP-Adressen auf logische Namen zurückschließen zu können. Man beachte allerdings die etwas gewöhnungsbedürftige Darstellung von IP-Adressen.

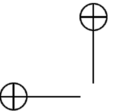
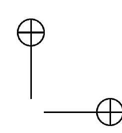
Bei dem folgenden Beispiel ist zu beachten, daß die Namen der Zonendatenbanken relativ zur Angabe `directory` in den globalen Optionen zu sehen sind. Ohne den symbolischen Link, den wir im *chroot*-Käfig angelegt haben, wäre es noch komplizierter. Dann müßten wir auch noch den Pfad des Käfigs voransetzen. So finden wir die Dateien unter */var/lib/named*.

Auch haben wir mit der Option `allow-update` sichergestellt, daß die Zonen nicht von anderen Rechnern aus mittels »Dynamic DNS Update« verändert werden können.

```
# Adressen der Art "<Name>.loopback"

zone "loopback" IN {
        type master;
        file "localhost.zone";
        allow-update { none; };
};
```


```
# Adressen der Art "127.0.0.<Nummer>"

zone "0.0.127.in-addr.arpa" IN {
        type master;
        file "127.0.0.zone";
        allow-update { none; };
};
```

`none` ist hierbei ein vordefinierter Name für eine Adreßspezifikation, die auf keinen Rechner paßt.

Wollen wir mit unserem DNS-Server noch weitere Zonen verwalten, brauchen wir nur neue `zone`-Einträge hinzuzufügen und neue Zonendatenbanken zu schreiben. Letzteres kann leicht geschehen, wenn wir die nun folgenden Beispiele als Ausgangsbasis nehmen.

Da wäre als erstes die Datenbank für das Netz *loopback*, die in der Datei */var/lib/named/localhost.zone* beschrieben wird:

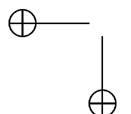
```
$TTL 86400
@           IN SOA localhost.loopback. root.localhost.loopback.   (
                            42              ; serial
                            3H              ; refresh
                            15M             ; retry
                            1W              ; expiry
                            1D )            ; minimum ttl
            IN NS           localhost
            IN MX 10        localhost
localhost   IN A            127.0.0.1
www         IN CNAME        localhost
```

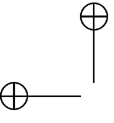
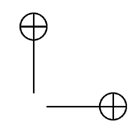
Am Anfang der Datei steht die Anweisung `$TTL 86400`. Damit wird die Standardzeit festgelegt, für die die folgenden Einträge von Klienten zwischengespeichert werden können. Nach Ablauf der angegebenen Zeit in Sekunden muß der Klient den Server neu fragen. Es ist allerdings möglich, für einzelne Einträge gezielt abweichende Werte anzugeben.

Unsere Zone ist ziemlich klein und hat nur fünf Elemente, die *Resource Records* (RRs) genannt werden. Der erste Record vom Typ SOA ist administrativer Natur. Sein erstes Element gibt den Ursprung der definierten Zone an. Hier müßte eigentlich `localhost.loopback.` stehen. Da wir den Ursprung der Zone aber schon in */etc[/bind]/named.conf* angegeben haben, können wir diese Angabe mit »@« abkürzen. Als nächstes Element gibt `IN` an, daß es um einen Eintrag der Klasse `Internet` geht. Es existieren auch die Klassen `Chaos` und `Hesiod`, die uns im Rahmen unserer Betrachtungen aber nicht interessieren. `SOA` bedeutet »Start of Authority«. Jede Datei darf nur einen `SOA`-Record enthalten. Dies muß darüber hinaus der erste RR sein. *localhost.loopback.* ist derjenige Server, auf dem die Daten für die Zone verwaltet werden.

`root.localhost.loopback.` steht für die E-Mail-Adresse des Verantwortlichen für die Zone (*root@localhost.loopback*). Dabei sollte beachtet werden, daß alle Angaben, die nicht auf ».« enden, als relativ zum Ursprung der Zone verstanden werden. Damit würde `localhost.loopback` ohne Punkt zu `localhost.loopback.loopback`.

Nun folgt eine Seriennummer, an der sekundäre Server erkennen können, ob die Daten der Zone geändert wurden. Sie sollte jedesmal aktualisiert werden, wenn dies geschieht.





Ihr folgen Angaben, die regeln, wie häufig sekundäre DNS-Server ihre Datenbank mit dem primären Server abgleichen. Prinzipiell erfolgt die Angabe in Sekunden, es ist aber auch möglich, sie in Stunden (H), Minuten (M) Tagen (D) und Wochen (W) anzugeben. Die erste Angabe »3H« gibt an, wie häufig sekundäre Server überprüfen sollen, ob sich die Seriennummer geändert hat. Die zweite Angabe »15M« regelt, in welchen Abständen es der sekundäre Server noch einmal probieren soll, wenn eine Abfrage der Seriennummer fehlschlägt. Gelingt es dem sekundären Server schließlich über einen Zeitraum von einer Woche (1W) nicht, sich abzugleichen, so hat er anzunehmen, daß seine Daten veraltet sind, und die Zone aus seiner Datenbank zu löschen.

Als letztes gibt »1D« die Zeit an, für die Klienten die Antwort zwischenspeichern dürfen, daß dem Server eine angegebene Domäne unbekannt ist (negative caching ttl).

Nun folgt die Angabe eine DNS-Servers für die Zone (NS). Prinzipiell können mehrere solcher Einträge vorhanden sein. Als nächstes wird angegeben, welcher Rechner E-Mails für die Zone annimmt (MX). Die Angabe »10« gibt die Priorität des Servers an. Je niedriger dieser Wert ist, um so höher ist die Priorität des Servers. Soll nun eine E-Mail zugestellt werden, so wird zuerst versucht, den Rechner mit der höchsten Priorität zu erreichen.

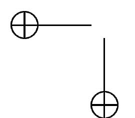
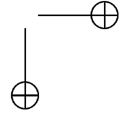
Als letztes folgen die Rechner der Zone. In diesem Fall existieren »localhost« und »www«, wobei www nur ein anderer Name für localhost ist. Zwei Punkte sind dabei zu beachten. Zum einen wird im A-Record zum ersten Mal eine IP-Adresse in numerischer Form angegeben. Diese endet nicht auf einem Punkt. Zum anderen ist die Benutzung von CNAME-Records umstritten. Alternativ könnte man auch einen weiteren A-Record verwenden. Will man aber nicht auf CNAME-Records verzichten, so sollte man beachten, daß ein Name, der als CNAME definiert wurde, niemals auf der rechten Seite eines SOA-, MX- oder CNAME-Records erscheinen sollte.

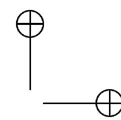
Es existieren noch weitere RRs. So erlaubt es beispielsweise ein HINFO-Record, die Hardware und das Betriebssystem eines Rechners anzugeben, während mit TXT-Records beliebige Zeichenketten mit dem Rechner verknüpft werden können. Allerdings sollte man sich überlegen, ob man einem potentiellen Angreifer diese Angaben tatsächlich frei Haus liefern will.

Um jetzt auch Anfragen nach IP-Adressen zu Namen auflösen zu können, wird zusätzlich die Zone 127.0.0 verwaltet. Allerdings ist dabei zu beachten, daß IP-Adressen der Form w.x.y.z für DNS als z.y.x.w.in-addr.arpa. angegeben werden (außer in der rechten Seite von A-Einträgen). Die Zone 127.0.0 wird in der Datei */var/lib/named/127.0.0.zone* definiert:

```
$TTL 86400
@   IN SOA      localhost.loopback. root.localhost.loopback. (
                42              ; serial
                3H              ; refresh
                15M             ; retry
                1W              ; expiry
                1D )            ; minimum

    IN NS       localhost.loopback.
1   IN PTR      localhost.loopback.
```



Es ist auch zu beachten, daß der Eintrag 1 (=127.0.0.1) nicht als »authoritative« (A), sondern nur als »Pointer« (PTR) eingetragen wird.

Konfiguration des Dienstaufrufs

Bevor wir den Dienst starten, müssen wir noch sicherstellen, daß er im `chroot`-Käfig läuft und er mit den Rechten des Benutzers `named` und nicht mit Rootrechten ausgeführt wird.

Dazu muß unter SuSE die Datei */etc/sysconfig/named* die folgende Zeile enthalten:

```
NAMED_RUN_CHROOTED="yes"
```

Bei anderen Distributionen müssen wir sicherstellen, daß der Dienst mit den folgenden Optionen gestartet wird:

```
-u named -t /var/lib/named
```

Die Option `-u` gibt dabei den Benutzer an, mit dessen Rechten der Dienst gestartet werden soll, während `-t` das Verzeichnis angibt, das als `chroot`-Käfig dient.

Um diese Optionen anzugeben, müssen wir als erstes das Runlevel-Skript finden, das den Dienst startet. Dies geschieht auf die bewährte Weise, dabei müssen wir allerdings beachten, daß das Programm, das wir suchen, nicht `bind`, sondern `named` heißt:

```
# find /etc/init.d -type f -exec grep -Hn named \{\} \;
/etc/init.d/bind9:5:# for a chrooted server: "-u nobody -t /var/lib/named"
/etc/init.d/bind9:6:OPTS=""
/etc/init.d/bind9:8:test -x /usr/sbin/named || exit 0
/etc/init.d/bind9:12:   echo -n "Starting domain name service: named"
/etc/init.d/bind9:14:       --pidfile /var/run/named.pid --exec /usr/sbin/named -- $OPTS
/etc/init.d/bind9:19:   echo -n "Stopping domain name service: named"
```

Unter Debian 3.1 heißt das Skript, das wir suchen, also `/etc/init.d/bind9`. Wenn wir uns das Skript genauer ansehen, stellen wir fest, daß das Programm nicht mit einem Parameter `-u` oder `-t` aufgerufen wird:

```
start-stop-daemon --start --quiet \
   --pidfile /var/run/named.pid --exec /usr/sbin/named -- $OPTS
```

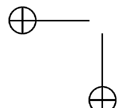
Allerdings wird eine Variable *OPTS* übergeben, in die wir zusätzliche Parameter eintragen können. Diese sollten wir aber nicht direkt hier ändern, sondern in der Datei */etc/default/bind9*. Diese Datei wird vom Runlevel-Skript eingebunden. In ihr definierte Variablen überschreiben die Variablen des Runlevel-Skripts.

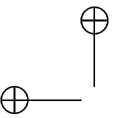
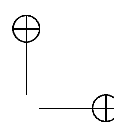
Derzeit ist in ihm nur eine Zeile enthalten, die den Inhalt der Variable *OPTS* definiert:

```
OPTS="-u bind"
```

Es ist also schon festgelegt, unter welchem Benutzerkonto der Dienst läuft. Nun müssen wir nur noch die Angabe unseres `chroot`-Käfigs hinzufügen:

```
OPTS="-u named -t /var/lib/named"
```

Start und Test

Den Server können Sie unter SuSE mit dem folgenden Befehl starten:

```
# /etc/init.d/named start
```

Damit der Dienst auch in Zukunft automatisch gestartet wird, sollten Sie den Dienst mit dem folgenden Befehl aktivieren:

```
# insserv named
```

Unter Debian ist der Dienst bereits gestartet. Sie können ihn aber mit

```
# /etc/init.d/bind9 restart
```

neu starten, um sicherzustellen, daß er auch wirklich die neue Konfiguration benutzt.

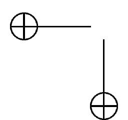
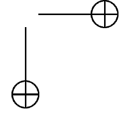
Nun sollten wir als erstes überprüfen, unter welcher Benutzerkennung der Dienst gestartet wurde:

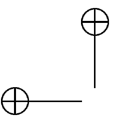
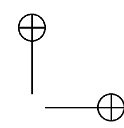
```
# ps aux | grep named
root   164  0.0  0.8  2040  496 ?      Ss  Nov01  0:03 /sbin/syslogd
named 1542  0.0  3.6 10192 2224 ?      Ss  00:16  0:00 /usr/sbin/named
named 1543  0.0  3.6 10192 2224 ?      S   00:16  0:00 /usr/sbin/named
named 1544  0.0  3.6 10192 2224 ?      S   00:16  0:00 /usr/sbin/named
named 1545  0.0  3.6 10192 2224 ?      S   00:16  0:00 /usr/sbin/named
named 1546  0.0  3.6 10192 2224 ?      S   00:16  0:00 /usr/sbin/named
root  1677  0.0  0.7  1340  472 pts/3  S+  00:37  0:00 grep named
```

Wir sehen, daß der Dienst mehrere Instanzen seiner selbst gestartet hat, die mit den Rechten des Benutzers named laufen. Nun müssen wir nur noch kontrollieren, ob auch ein chroot durchgeführt wurde.

Dazu brauchen wir die Prozeßnummer der Instanz, die wir untersuchen wollen. Hier bietet sich z. B. 1542 an. Wir finden die gesuchte Information im entsprechenden Verzeichnis des Prozesses:

```
# ls -l /proc/1542
total 0
dr-xr-xr-x  2 root  root  0 Nov  2 00:47 attr
-r--------  1 root  root  0 Nov  2 00:47 auxv
-r--r--r--  1 root  root  0 Nov  2 00:35 cmdline
lrwxrwxrwx  1 root  root  0 Nov  2 00:47 cwd -> /var/lib/named
-r--------  1 root  root  0 Nov  2 00:47 environ
lrwxrwxrwx  1 root  root  0 Nov  2 00:47 exe -> /usr/sbin/named
dr-x------  2 root  root  0 Nov  2 00:47 fd
-r--r--r--  1 root  root  0 Nov  2 00:47 maps
-rw-------  1 root  root  0 Nov  2 00:47 mem
-r--r--r--  1 root  root  0 Nov  2 00:47 mounts
lrwxrwxrwx  1 root  root  0 Nov  2 00:47 root -> /var/lib/named
-r--r--r--  1 root  root  0 Nov  2 00:35 stat
-r--r--r--  1 root  root  0 Nov  2 00:47 statm
-r--r--r--  1 root  root  0 Nov  2 00:35 status
dr-xr-xr-x  3 root  root  0 Nov  2 00:47 task
-r--r--r--  1 root  root  0 Nov  2 00:35 wchan
```

Der symbolische Link *root* zeigt auf das Wurzelverzeichnis des Prozesses. Normalerweise wäre dies /. Daß hier etwas anderes steht, bedeutet, daß ein `chroot`-Aufruf durchgeführt wurde.

Nun sollten wir die Konfiguration des Nameservers überprüfen. Eine einfache Methode hierzu besteht darin, sich mit `dig` den Inhalt der einzelnen Zonen anzeigen zu lassen. Für die Zone »loopback« lautet der Aufruf:

```
> dig @127.0.0.1 loopback axfr
```

Hierbei gibt das erste Argument den zu verwendenden Nameserver, das zweite die nachzufragende Adresse oder Domain und das dritte schließlich die Art der anzuzeigenden RRs an. »axfr« heißt hierbei, daß wir einen Zone Transfer wünschen. Andere gültige Werte wären z. B. ANY (alle relevanten RRs) sowie die schon oben aufgeführten Typen von RRs (A, MX, NS, . . .)[11].

Die Ausgabe sollte etwa folgendermaßen aussehen.

```
; <<>> DiG 9.2.1 <<>> @127.0.0.1 loopback axfr
;; global options:  printcmd
loopback.              86400   IN      SOA     localhost.loopback. \
root.localhost.loopback. 42 10800 900 604800 86400
loopback.              86400   IN      NS      localhost.loopback.
loopback.              86400   IN      MX      10 localhost.loopback.
localhost.loopback.    86400   IN      A       127.0.0.1
www.loopback.          86400   IN      CNAME   localhost.loopback.
loopback.              86400   IN      SOA     localhost.loopback. \
root.localhost.loopback. 42 10800 900 604800 86400
;; Query time: 470 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Nov  4 00:04:14 2004
;; XFR size: 7 records
```

Daß hierbei der SOA-RR doppelt erscheint, ist normal. Dies ist in den relevanten RFCs für Zone Transfers so vorgeschrieben.

Nachdem wir den Vorgang für die Zone »0.0.127.in-addr.arpa« wiederholt haben, können wir sicher sein, daß wir unsere Zonen richtig aufgesetzt haben.

11 Die Groß- und Kleinschreibung spielt hierbei keine Rolle. Auch SoA wäre ein gültiges Argument.

